NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

REALISTIC TRAFFIC GENERATION CAPABILITY FOR SAAM TESTBED

by

Fatih Turksoyu

March 2001

Thesis Advisor:	Geoffrey Xie
Second Reader:	Bert Lundy

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 2001	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE: Realistic Traffic Generation Capability For SAAM Testbed	5. FUNDING NUMBERS
6. AUTHOR(S) Turksoyu, Fatih	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA and NASA	10. SPONSORING / MONITORING AGENCY REPORT NUMBER G417

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE Statement A

13. ABSTRACT *(maximum 200 words)*

Traffic modeling is an important component of the design of any communication network. This is even more crucial for emerging networks, which are expected to operate in high speed and high bandwidth environments. As the design of a network depends to a great extent on the types of traffic it is expected to carry, it is essential to characterize the traffic that a network expected to carry. This is where traffic models are very important. They can be used to produce artificial traffic input that exhibit essential characteristics of real network loads.

This thesis describes a design and implementation of a general-purpose traffic generator for a test bed of the Server and Agent Based Active Network Management (SAAM) architecture. The traffic generator is easy to use and implements the four most important traffic models (Constant Bit Rate (CBR), Poisson, Packet-Train, and Self-Similar). With this traffic generator, the SAAM project now has the capability of simulating and testing the system components in more accurate and more realistic environments.

14. SUBJECT TERMS Traffic Model, CBR, Poisson, Packet-Train, Self-Similar, Next Generation Internet, Networks	15. NUMBER OF PAGES 172
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-)
Prescribed by ANSI Std. 239-18298-102

i

THIS PAGE INTENTIONALLY LEFT BLANK

REALISTIC TRAFFIC GENERATION CAPABILITY FOR SAAM TESTBED

Fatih Turksoyu
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
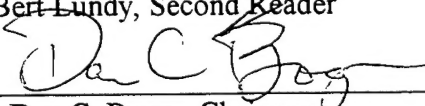**March 2001**

Author: _____

Fatih Turksoyu

Approved by: _____

Geoffrey Xie, Thesis Advisor

_____

Bert Lundy, Second Reader

_____

Dan C. Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Traffic modeling is an important component of the design of any communication network. This is even more crucial for emerging networks, which are expected to operate in high speed and high bandwidth environments. As the design of a network depends to a great extent on the types of traffic it is expected to carry, it is essential to characterize the traffic that a network is expected to carry. This is where traffic models are very important. They can be used to produce artificial traffic input that exhibits essential characteristics of real network loads.

This thesis describes a design and implementation of a general-purpose traffic generator for a test bed of the Server and Agent Based Active Network Management (SAAM) architecture. The traffic generator is easy to use and implements the four most important traffic models (Constant Bit Rate (CBR), Poisson, Packet-Train, and Self-Similar). With this traffic generator, the SAAM project now has the capability of simulating and testing the system components in more accurate and more realistic environments.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to extend my sincere gratitude to my thesis advisor, Professor Geoffrey Xie and Professor Bert Ludy, for their patience. I would also like to thank to my parents, whose love and patience have been a constant inspiration to me throughout my two years of study at the Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A.   BACKGROUND

The Internet started as a research project in the late 60's, has grown rapidly in the last couple

of years and became a worldwide information network. Nobody knows the precise number of users

connected to the Internet today. But there are statistics that show the volume of traffic on the

Internet grows exponentially. A conservative estimate of Internet traffic growth is that it doubles

every 6 months. With this growth rate the aggregate bandwidth required for the Internet in the

United States alone will be about 35 Tbps (Terabits per seconds) by 2001-2002 [Ref.1]. The

characteristics of the Internet traffic also have changed significantly in the last few years. Five to

seven years ago, the Internet was mainly used for file transfer, e-mail, and access to the World Wide

Web. Nowadays the voice and video traffics are also common in Internet.

The increase in volume and variety of the network traffic raises new challenges for network

management. In order to support applications as diverse as teleconferencing, video-on-demand, e-

commerce and distributed computing, the resource management of the Internet needs to be

improved.

These challenges have led people to look for solutions and new designs that can support a

range of Quality-of-Service (QoS) based on application requirements. The Internet routing and

protocol system must be redesigned to support QoS. Some people call this effort the "Next

Generation Internet" initiative (NGI) or "Internet version 2".

Several QoS models [Ref. 2] have been developed recently. The first one is Integrated

Service (IntServ), which is characterized by resource reservation. The second one is Differentiated

1

Service (DiffServ). The third is Multi-Protocol Label Switching (MPLS). The last two are characterized by relative QoS.

B.   SERVER AND AGENT BASED ACTIVE NETWORK MANAGEMENT

Under the DARPA-funded Next Generation Internet (NGI) initiative, a research project was initiated at Naval Postgraduate School in 1998 to develop a Server and Agent Based Active Network Management (SAAM) system that provides an efficient solution for QoS. The SAAM architecture is designed to scale well with integrated services.

1.   What is SAAM?

SAAM is a network management system that enables a network to provide integrated services. Instead of a totally router-based architecture, SAAM utilizes a server-based hierarchical routing architecture that provides QoS routing services for network resource intensive applications.

The goal of SAAM according to Varble and Yarger is to "find a solution that will provide a guaranteed QoS while still maintaining the simplicity and robustness of the underlying TCP/IP architecture." [Ref.4]. SAAM seeks to provide scalable and customizable QoS through multiple levels of services, centralized network management, and hierarchical routing.

The design of SAAM targets three types of network services:

a.   *Best Effort Service*

This type of service is the traditional Internet data service. There are no guarantees for a client to receive sufficient bandwidth, nor are there any guarantees that the client's packets will arrive at the destination within a specific time frame, or they will arrive at all. Delay sensitive applications such as teleconferencing would suffer from this lack of performance guarantees if they subscribe to the Best Effort service.

2

b.   Integrated Service

The Integrated Service, unlike the Best-Effort Service, provides guaranteed QoS that can be customized on a per client session basis. A client subscribing to this type of service is guaranteed to receive the quality of service (minimum bandwidth, bound on delay and loss rate, etc.) that he or she has negotiated for an application or session. Integrated Service is characterized by resource reservation. The client application must go through the resource reservation process and wait for the service provider to set up a transmission path and reserve resources before it can transmit packets. The resource reservation may be rejected if the service provider finds no resource available for handling the additional traffic; in that case the client application cannot transmit packets at all.

c.   Differentiated Service

The current Internet offers only a very simple quality of service (QoS), Best-Effort data delivery. Before real-time applications can be broadly used, the Internet infrastructure must be modified to support real-time QoS, which provides some control over end-to-end packet delays. IETF first proposed Integrated Service architecture, which uses RSVP to reserve network resources to assure the service quality. This approach has fundamental scaling problem in that per flow state is maintained at all routers and end-systems supporting a flow. Therefore, there has to be something that falls between the Best-Effort service of the current Internet and the envisioned per-flow heavyweight mechanisms of RSVP and IntServ. The answer is called Differentiated Service (DiffServ). The service accommodates heterogeneous application requirements and user expectations, permits differentiated pricing of Internet Service and at the same time achieves scalability.

3

In Differentiated Service architecture, complex classification and conditioning functions of traffic are implemented only at network boundary nodes and, within the core of the network, per-hop behaviors are applied to aggregates of traffic which have been appropriately marked using the DS field in the IPv4 or IPv6 header. Hence, per-application flow or per-customer forwarding state need not be maintained within the core of the network. Thus, it can solve the scaling problem in the Integrated Service model.

2. SAAM Architecture

SAAM architecture consists of a SAAM server that controls a SAAM region including a number of routers. The server will collect the global picture of the region and will make decisions on behalf of the routers. This method will provide a lightweight router that will perform only its primary task of forwarding the packets to their destination addresses. To make its service scalable for large networks, SAAM organizes its SAAM servers in a hierarchy. At the first level of the hierarchy, SAAM partitions the network into autonomous regions. These regions are called SAAM regions.

SAAM assigns one SAAM server for each SAAM region. Those regional servers report to higher level servers that have control over all the regional servers. This central management gives the SAAM system the advantage of having control over the entire network and enables the applications to have one point of reference to obtain QoS service. Instead of negotiating with local routers or regional servers for end-to-end service that they cannot guarantee, the service request is always sent to the suitable server that can make the correct decision for end-to-end service. Also, this design reduces the processing power requirements on the router side. Figure 1.1 illustrates the hierarchical structure of SAAM.

4
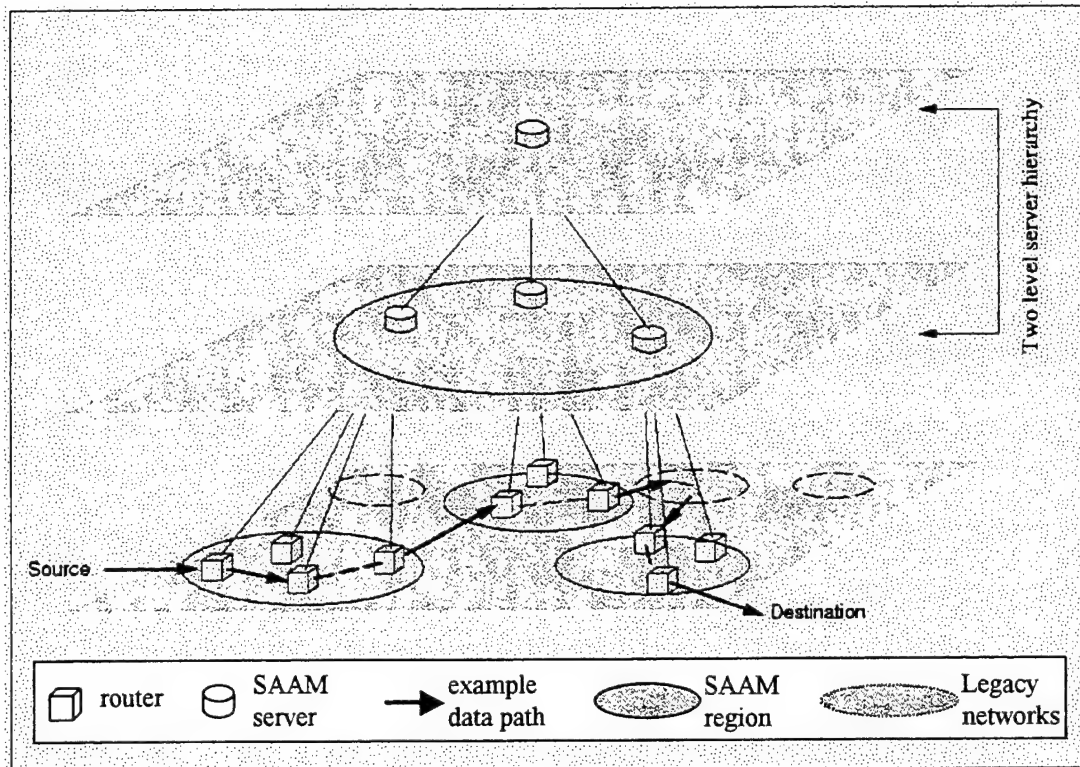
Figure 1.1 Hierarchical Organization of SAAM Servers [From Ref. 4]

There are two major components in SAAM:

a.   The SAAM Server

The SAAM server maintains an accurate picture of the QoS capabilities of the network by periodically retrieving link performance information from the routers, and aggregating this information into a ready-to-use database of useful paths. This database is called the *Path Information Base* (PIB) (shown in Figure 1.2). By using the PIB, the SAAM server can efficiently implement network functions such as QoS routing and re-routing of real-time flows, which are required for providing guaranteed and differentiated quality of services.

5
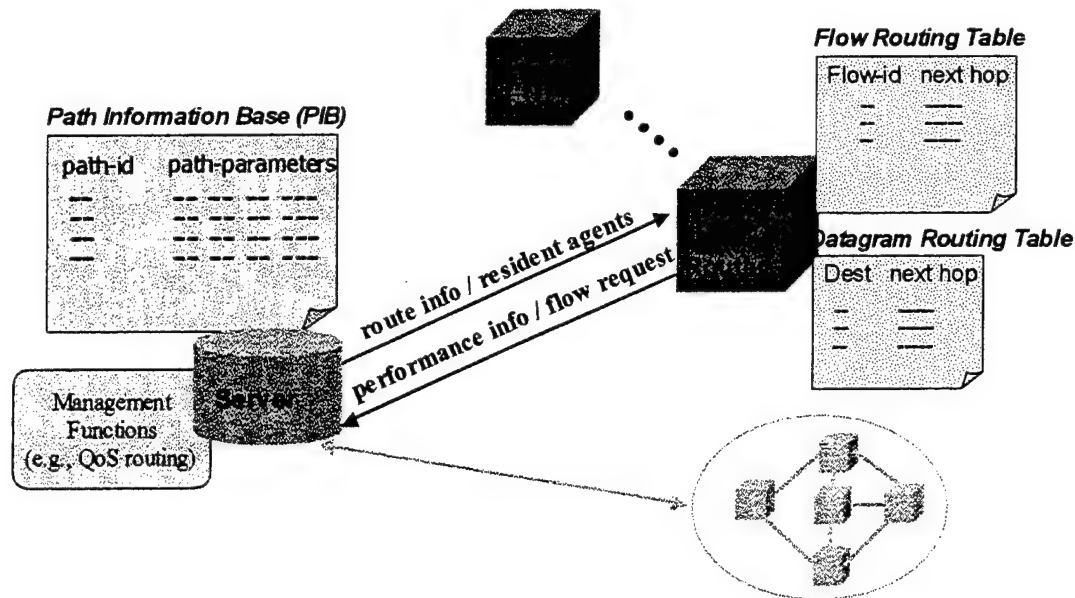
Figure 1.2 Logical Model of SAAM [From Ref. 4]

b.   *The SAAM Router*

The SAAM Routers handle actual data traffic from client applications. All client

resource reservation requests enter the network through the edge routers, which forwards the

requests to the server for approval. The server will respond to each request either by assigning a

path to meet the request or by denying the request. When a path is assigned to a client for the first

time, the server will also send route-update messages to all routers in the path to install appropriate

entries to their routing tables to create the packet-forwarding path.

Each router is responsible for reporting its status (i.e., the state of its interfaces) to

the server so that the server always has an accurate view of the network to make good routing

decisions. Due to the hierarchical structure of the SAAM system, that status information first goes

up to the regional server, then aggregated and forwarded by the regional server to servers higher in

6

the hierarchy. This reduces the traffic on the network for the purpose of the router status update, and keeps all decision makers informed of the status of each router.

The SAAM prototype is designed to support resident agents, allowing the components of a router to be upgraded and installed dynamically during runtime. The precompiled byte code of a resident agent is registered as a new module of the receiving node. The module may run itself by creating a thread, or an existing thread may call it. For instance, after a node in the prototype receives a server resident agent it becomes a SAAM server, otherwise it stands up as a router. This adds the flexibility to the system, so that the components of the server or the routers can be installed or uninstalled after the system is set and running according to the system needs at that moment.

c. *The SAAM DemoStation*

The DemoStation is the component that initializes a SAAM test topology using the Java-based SAAM prototype. It creates software-emulated SAAM servers and routers on target host or hosts, and sends additional agents to specific servers and routers.

SAAM test topologies are stored in XML files. The development of XML based SAAM test topology configuration is explained in detail by Abanneh [Ref. 5].

C. SCOPE OF THIS THESIS

The primary goal of this thesis is to explore the characteristics of four different network traffic models (Constant Bit Rate (CBR), Poisson, Packet-Train, and Self-Similar), and integrate these models into the SAAM prototype. By implementing these models, the SAAM project will have the capability of testing the system components for more accurate and more realistic traffic environments.

D.   ORGANIZATION OF THIS THESIS

The remainder of this thesis is organized into the following chapters:

- Chapter II: Overview of Traffic Models. Provides information about CBR Model, Poisson Model, Packet-Train Model, and Self-Similar Model.

- Chapter III: Design of Traffic Generator Agent. Explains the classes and methods added to the SAAM prototype to implement the four traffic models.

- Chapter IV: Implementation of Traffic Generator Agent. Explains the details of how the traffic models are integrated into the SAAM prototype.

- Chapter V: Tests and Results. Explains and shows how well the traffic agents function.

- Chapter VI: Conclusions. Summarizes the results from this thesis and outlines possible future work.

II.   OVERVIEW OF TRAFFIC MODELS

A.   INTRODUCTION

Understanding the models of network traffic will help us design better protocols, better network topologies, and better routing and switching hardware. It will help network operators provide better services to the users.

Characteristics of network traffic play crucial role in performance analysis and design of networks. One of the most important and challenging issues in the design of networks is to develop an efficient and integrated framework within which requested QoS is fully supported. The presence of different service requirements makes the design and management of networks very complex. A client application may require guarantees on various QoS parameters such as throughput, packet loss, delay and jitter. These guarantees represent performance objectives expected from the network for the entire duration of the connection.

Traffic modeling is an important component of the design of any communication network. This is even more crucial for emerging networks, which are expected to operate in high speed and high bandwidth environments. Rapid development of new technologies and services are expected to impact the growth of these networks as well. These technologies are expected to support many and diverse applications with different and sometimes conflicting traffic management requirements (such as no loss, high throughput for data vs. acceptable loss, small delays for video). As such, a broad range of QoS guarantees needs to be supported simultaneously; it is necessary to properly design flexible flow control and bandwidth allocation mechanisms. These mechanisms must be designed under realistic assumptions of network conditions including input traffic patterns.

As the design of a network depends to a great extent on the type of traffic it is expected to carry, it is essential to correctly characterize the traffic that the network is expected to carry.

This is where traffic models are very important. They can be used to produce artificial traffic input that exhibits essential characteristics of real network loads. Such artificial traffic input is needed in simulation studies or performance analysis that aim to validate new network protocols before their costly deployments. If the traffic models used do not accurately represent actual traffic, network performance may be overestimated or underestimated.

Each traffic model describes a parameterized stochastic process of packet generations. In general, a good traffic model is able to capture the characteristics of traffic from a particular class of applications with a minimum number of parameters.

B. TYPES OF TRAFFIC MODELS

Traffic models basically fall into two categories: (i) short range dependent models and (ii) long range dependent models [Ref. 19]. Both short and long range dependence refer to properties of wide sense stationary stochastic processes (i.e., time series which have a constant mean and a covariance function which depends only on the time difference.) Specifically, let $X = \{X_t; t = 0,1,2,....\}$ be a scalar process. We define its mean m and covariance function r by

$$m(t) \equiv E[X_t] \quad \text{and} \quad r(s,t) \equiv E[X_s X_t] - m(t)m(s), \quad s,t = 0,1,2... \tag{2.1}$$

The scalar process $X = \{X_t; t = 0,1,2,....\}$ is said to be **wide-sense stationary process** if

$$m(t) = m(0) \quad and \quad r(s;\ t) = r(|t - s|), \quad s,\ t = 0,\ 1,\ .... \tag{2.2}$$

In other words, the mean function is constant and the covariance function depends on the arguments s and t only through the difference $|t-s|$ [Ref. 19].

Short Range Dependent Models (SRD): The defining characteristic of this class of wide sense stationary models is the summability of the covariance function, i.e.,

$$\sum_{h=0}^{\infty} r(h) < \infty \qquad (2.3)$$

However, a lot of short-range dependent models satisfy a much stricter property, namely they have an exponentially decaying covariance function. This exponentially decaying covariance function implies that the lengths of packets generated at two time instants very far apart will not be correlated. The rate of the exponential decay can often be expressed as a function of the parameters defining the model. Classical models such as autoregressive, and Markov models are short-range dependent [Ref. 19].

Long Range Dependent Models (LRD): A wide-sense stationary stochastic process is said to be LRD if the covariance function is not summable,

$$r(h) = \sum_{h=0}^{\infty} r(h) \qquad (2.4)$$

This non-summability of the correlations captures the intuition behind long-range dependence, namely that through the high lag correlations might be individually small, their cumulative effect counts and gives rise to features, which are drastically different from SRD processes. Any process with a hyperbolically decaying covariance function, namely

$$r(k) \sim k^{-D} (k \to \infty) \text{ with } 0<D<1 \qquad (2.5)$$

11

satisfies this criterion. This hyperbolic decay, being much slower than an exponential decay, also emphasizes the notion of long-range dependence [Ref. 19]. Two packets generated at two time instants very far apart may still have a considerable amount of correlation. Examples of long-range dependent models are the Fractional Gaussian Noise model and the model based on the $M/G/\infty$ queue [Ref. 19].

Stochastic models of packet traffic used in past were exclusively Markovian in nature, or more generally short-range dependent traffic processes. Those traffic models, now called classic models, typically assumed Poisson arrival rate and exponentially distributed message sizes. Data source models with those characteristics were used in the analysis and modeling of early ARPANET and agreement between real data and results from queuing models were good and satisfactory [Ref. 19]. And so were agreements in 10-15 years that followed. Traditional telephony has benefited very much (for understanding its internal behavior and for a system design) for a long time (more than a half century) from classic traffic models, which basically assume that call holding times are exponentially distributed [Ref. 12]. But, recent studies [Ref. 13] have shown that call holding times may be best described using heavy-tailed distributions with possibly infinite variance/mean. These characteristics are contrary to those of exponential distributions. One possible reason for the change in characteristics of telephone traffic is that telephony systems are now being used not only for its traditional voice communication but also more and more for computer data communication.

Apparently those new traffic mixes have quite different characteristics than voice data alone and as their share of overall traffic has grown, so overall traffic characteristics have diverted from the classic models.

Many studies indicate considerable increase in overall amount of traffic in communication networks. But, noticeable are also new types of traffic generated by new network applications such as World Wide Web, Gopher and newsgroups, which are quite different from traditional applications such as file transfer protocol (FTP), remote access (telnet) and E-mail (smtp). These new types of traffic can obviously change the overall traffic characteristics in the networks. As result of these observations and trends, since mid 1980's, research in the area of traffic characterization and its implications on design of computer networks has intensified. The model of "packet trains" was first introduced in 1986 [Ref. 15] and has been widely used to generate artificial compressed video or audio traffic. This model assumes that a group of packets travel together as a train, contrary to the Poisson model, which assumes that packets are independent. In comparison, Poisson can be seen as "car model". Another feature of today's networking traffic was discovered in the mid 90's. Specifically, long-range dependency (long-memory) was found in real traffic traces from both local area networks and wide-area networks [Ref. 16].

This thesis focuses on four traffic models that are most widely used. They are: the constant bit rate (CBR), Poisson, Packet-Train and Self-Similar models. Rest of this chapter, these models will be explained.

C. CBR TRAFFIC MODEL

Audio traffic and old video codecs (coder/decoder) inject constant bit rate traffic into networks. These applications cannot function with less bandwidth than some minimum application specific requirement. They do not benefit from extra bandwidth either.

13

The Constant Bit Rate (CBR) model is used to simulate traffic that has characteristics of uncompressed voice and video streams in real networks. A CBR traffic source produces a continuous sequence of fixed size packets that are evenly spaced in time.

This model is very easy to implement, because it only uses one parameter to generate traffic. The parameter is the packet rate (R), which represents the number of packets that must be generated in specific time.

The formula 2.6 shows the simplicity of the CBR model.

$$I = \frac{1}{R} \tag{2.6}$$

The result I is used as a time difference between each packet generation. For example, if 10 packets are to be generated each second, the time difference between each packet will be

$$I = \frac{1}{10} = 0.1 \text{ sec} = 100 \text{ } ms$$

So the generator will generate one packet each 100 ms.

D. POISSON TRAFFIC MODEL

Poisson model is the oldest traffic model and a great deal of earlier work has been done in analyzing networks using this model. Traffic loads in Ethernet LANs were often modeled as Poisson processes because they have attractive theoretical properties and are well suited for classic computer applications (FTP, Telnet, and Email). First studies on data traffic [Ref. 18] indicated that the data traffic sources in communication networks were often bursty in nature, that is, relatively short sequence of source activities are followed by long idle periods. During 70's and early 80's, some other studies suggested the following assumptions for external data sources [Ref. 12].

14

The inter-arrival times of messages generated by an external data source are exponentially distributed. Each external data source behaves as a Poisson process. Let $G(i)$, $i = 1,2,3,.....,N$ be a random variable denoting inter-arrival times of messages generated from the ith data source.

The length of messages generated by an external data source is exponentially distributed. Let $H(i)$, $i = 1,2,3,.....,N$ be a random variable denoting length of messages from the i th data source. Processes described by random variables $G(i)$ and $H(i)$ are stationary and independent.

As a consequence of the first assumption, the aggregate traffic from several data sources would get smoother and smoother with an increase of the number of sources [Ref. 12].

The assumption about exponential distribution of message sizes can be relaxed to use a general distribution, and still closed-form solutions for different statistical parameters (mean, variance and other moments) could be obtained using queuing theory methods.

The Poisson traffic model is a very common traffic model in network simulations. The reason for this is based on several facts [Ref. 18]:

The Poisson distribution is still accurate for describing traffic from many network applications. Much real traffic possesses the property of Poisson distribution and can be approximately simulated with a Poisson sequence.

The Poisson distribution had been widely studied for a very long period. It is very suited for comparative studies.

The generation of Poisson sequence is very easy. With modern computer technology, one can develop a Poisson sequence generator with several lines of high-level programming code.

The Poisson model source creates fixed-size packets according to a Poisson process. In other words, a pseudo-random number generator that draws from an exponential distribution determines the amount of time that elapses between each packet generation. Given a pseudo-random generator that chooses a real number U from a uniform distribution in the range [0.0, 1.0), a pseudo-random number I, the current packet inter-arrival time, can be determined from an exponential distribution as follows:

$$I = \frac{\ln(1 - U)}{-R} \tag{2.7}$$

Where $1/R$ is the desired mean (average inter-arrival time) of the exponential distribution (R is the average packet rate).

E.   PACKET-TRAIN TRAFFIC MODEL

Traditionally, communication networks have been modeled with either Poisson or compound Poisson models. A study of a token ring local area network (LAN) at MIT [Ref. 15] found that packet arrival followed neither of these models. Instead, traffic followed a more general model named the "packet train," which describes network traffic as a collection of packet streams traveling between pairs of nodes. A packet train consists of a number of packets traveling between a particular node pair.

The packet train model assume that a group of packets travel together, and it should be obvious that a protocol design based on the assumption of a train arrival would be quite different from one based on independent arrivals. In the car model, each car has to decide at each intersection (or exit) whether to take exit or not. Even if all packets are going to one destination, they each make independent decision, which may result in unnecessary overhead. The overhead is apparent on

computer networks in which all intermediate nodes (routers, gateways, or bridges) must make this decision for all packets. In a train model, on the other hand, the locomotive (the first packet of the train) may make the routing decision, and all other packets of a train may follow it. It must be pointed out that the packet train model is a source model. It applies only when we look at the packets coming or going to a single node. Unlike the Poisson processes, trains are not additive. The sum of a number of trains is not a train. The source can be either in generation (train) state or idle (inter-train) state. The transitions between these states are memoryless (Markovian). The duration of the two states is exponentially distributed, with inter-train arrival times usually of the order of several seconds and inter-car times inside the trains of the order of a few milliseconds [Ref. 13]. Figure 2.1 shows the inter-car and inter-train times.
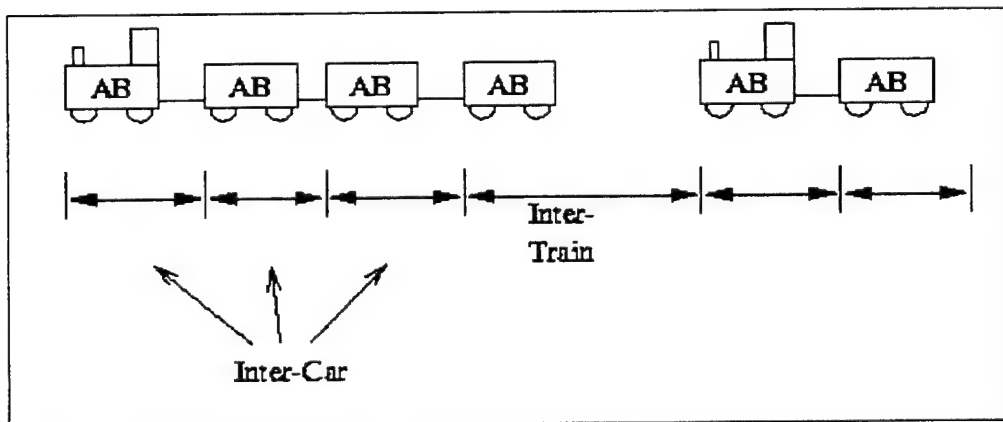
Figure 2.1 Times in Packet-Train Model [From Ref. 15]

This model reflects the fact that much of the communication inside a network involves in fact many packets spaced closely in time that are exchanged between the same two endpoints. More

17

implementation details of this model, including the formulas we use to implement this model in the SAAM prototype, will be explained in the next chapters.

F. SELF-SIMILAR TRAFFIC MODEL

1. Self-Similarity in Network Traffic

Self-Similarity in network traffic was stated in 1993 with the publication of the paper titled "On the Self-Similar Nature of Ethernet Traffic" [Ref. 7]. A self-similar phenomenon displays structural similarities across a wide range of timescales. Traffic that is bursty on many or all timescales can be described statistically using the notion of self-similarity. Self-similarity is the property associated with "fractals," which are objects whose appearances are unchanged regardless of the scale at which they are viewed [Ref. 7].

The authors reported the results of a massive study of Ethernet traffic and demonstrated that it had a self-similar (i.e., fractal) characteristic. This meant the traffic had similar statistical properties at a range of timescales: milliseconds, seconds, minutes, hours, even days and weeks. Another consequence is that the merging of traffic streams, as in a statistical multiplexer or an asynchronous transfer mode (ATM) switch, does not result in smoothing of traffic. Again, bursty data streams that are multiplexed tend to produce a bursty aggregate stream. The results show the self-similarity in ATM traffic, compressed digital video streams, and Web traffic between browsers and servers. Although a number of researchers had observed over the years that network traffic didn't always obey Poisson assumptions used in queuing analysis, this paper for the first time provided an explanation and a systematic approach to modeling realistic data traffic patterns.

2. Theoretical Background

Babic, Valdalore and Jain [Ref. 12], explained the self-Similarity and its properties as follow:

18

For a stochastic process $X = (X_t : t = 0,1,2....)$ to be a second order (weak or covariance or wide-sense) stationary, it is sufficient to have the existence of a stationary mean $\mu = E[X_t]$, a stationary and finite variance

$v = E[(X_t - \mu)^2]$, and a stationary auto-covariance (function) associated with a process of observations made at successive times $k = \text{cov}(X_t, X_{t+k}) = E[(X_t - \mu)(X_{t+k} - \mu)], k = 0,1,2,......$, that depends only on k and not on t.

Let $X = (X_t : t = 0,1,2....)$ be a second order stationary stochastic process, with a mean $\mu = E[X_t]$, a variance $v = E[(X_t - \mu)^2]$, and auto-covariance (function) $\gamma_k = \text{cov}(X_t, X_{t+k}) = E[(X_t - \mu)(X_{t+k} - \mu)], k = 0,1,2,......$, Note, $v = \gamma_0$. Let the auto-correlation (function) of X at lag k be denoted as ρ_k, and by definition $\rho_k = \gamma_k / \gamma_0$.

We can think of a packet traffic process X consisting of a set $\{X_t\}$, where X_t is the number of packets that arrive in the t-th time unit.

Let $X^m = (X_j^{(m)} : j = 1,2,3,...)$ for each $m = 1, 2, 3, ...$, be the new second order stationary process, obtained by averaging the original process X over non-overlapping blocks of size m.

$X_j^{(m)} = (X_{jm-m+1} + X_{jm-m+2} + ... + X_{jm})/m$, with the variance v_m, the auto-covariance $\gamma_k^{(m)}$, and the autocorrelation $\rho_k^{(m)}$. It can be shown that:

$$v_m = v/m + 2/m^2 \sum_{k=1}^{m} (m - k)\gamma_k \tag{2.8}$$

or

19

$$v_m = v/m + 2/m^2 \sum_{s=1}^{m-1} \sum_{k=1}^{s} \gamma_k \tag{2.9}$$

The stochastic process X is said to have short-range dependency if $\sum_k \gamma_k$ is convergent,

Equivalently, from equation (2.9)

$$v_m \approx v^l/m \quad \text{with } v^l \text{ finite, for large m} \tag{2.10}$$

Such process is also called a stationary process with short memory or short-range correlations or weak dependence. An example of a stationary short-range dependent process would be a stochastic process with exponentially decaying auto-covariance function, i.e.

$$\gamma_k \approx r^l a^k \quad \text{for large k, } 0<a<1$$

Assuming 3 holds, it can be shown that $\gamma_k^{(m)} \to 0$ for $k=1,2,\ldots,$for large m. Then, it can be concluded that the aggregated (averaged) processes $X^{(m)}$, derived from the short-range dependent process X, for large m tend to covariance (second order) stationary white (pure) noise.

A stochastic process X is said to have long-range dependency if $\sum_k \gamma_k$ is divergent, i.e. $\sum_k \gamma_k \to \infty$. Equivalently, from 2.8

$$mv_m \to \infty, \quad \text{for large } m$$

Such process is also called a stationary process with long memory or long-range correlations or strong dependence. An example of a stationary long- range dependent process would be a stochastic process with hyperbolically decaying auto-covariance function.

$$\gamma_k \approx r^l k^{-\beta}, \text{ for large } k, 0 < \beta < 1$$

or equivalently

20

$$v_m \approx v^l m^{-\beta} \text{ for large } m, 0 < \beta < 1$$

From an intuitive point, possibly the most enlightening property is that the averaged process $X^{(m)}$ takes a nondegenerated correlation structure for large m. An implication is that the averaged process $X^{(m)}$ will not appear as white noise. Instead, the (typical) aggregated traffic will have bursty sub periods and less bursty sub periods for small as well large time-scales.

It can be shown that for long-range dependent processes

$$\rho_k^{(m)} \rightarrow \rho_k \quad \text{for large k and m} \tag{2.11}$$

Equation 2.11 indicates that for k and m large enough, auto-correlation does not depend on m, but only on k. This property is called asymptotic second-order self-similarity. So, long-range dependency implies asymptotic second-order self-similarity.

The process X is said to be exactly second-order self-similar (or fractal) if

$$\rho_k^{(m)} \rightarrow \rho_k \text{ for all } m, k >= 0$$

and

$$v_m = vm^{-\beta} \text{ for all } m, k >= 0$$

Above implies that the process X and the averaged processes $X^{(m)}$ have identical correlational structure and "look" alike. Usually, instead of the parameter β, the parameter $H = 1 - \beta/2$ is used and it is called Hurst coefficient. H (Hurst) coefficient characterizes the stochastic processed as follows:

for $1/2 < H < 1$, the process has long-range dependence,

for $H <= 1/2$, this is the case of a process with short-range dependence or independence.

The Hurst Parameter: The Measure of Self-Similarity

The Hurst parameter H is a measure of the level of self-similarity of a time series. H takes values from 0.5 to 1. In order to determine if a given series exhibits self-similarity, a method is needed to estimate H for a given series. There are several approaches to doing that [Ref.20], some of them are :

Analysis of the variances of the aggregated processes $X^{(m)}$

Analysis of the rescaled range *(R/S)* statistic for different block sizes

A Whittle estimator

The first method, the variance time plot, relies on the slowly decaying variance of a self-similar series. The variance of $X^{(m)}$ is plotted against m on a log-log plot. Then a straight line with a slope $(-\beta)$ greater than -1 is indicative of self-similarity and the parameter H is given as above [Ref .6].

The second method, the *R/S* plot, uses the fact that for self-similar data, the rescaled range or *R/S* statistic grows according to a power law with exponent H as a function of the number of points included, n. Thus, the plot of *R/S* against n on a log-log plot has a slope which is an estimate of *H*.

While the preceding two graphical methods are useful to estimate H, they may be biased for large H. The third method, a Whittle estimator, does provide a confidence interval. This technique uses the property that any long-range dependent process approaches Fractional Gaussian noise (FGN) when aggregated to a certain level, and so should be coupled with a test of the marginal distribution of the aggregated observations to ensure that it has converged to the normal distribution [Ref. 6]. As m increases, short-range dependences are averaged out of the data set.

If the value of H remains relatively constant, it is almost certain that this H value measures a true level of self-similarity of the data set.

3.   Causes of Self-Similarity

Since self-similarity is believed to have a significant impact on network performance, understanding the causes of self-similarity in traffic is important. Research done by M. E. Crovella [Ref. 6] has revealed that the traffic generated by World Wide Web transfers shows self-similar characteristics. Comparing the distributions of ON and OFF times, they found that the ON time distribution was heavier-tailed than the OFF time distribution. The distribution of file sizes in the Web might be the primary determiner of Web traffic self-similarity. In fact, the work presented by K. Park [Ref. 10] has shown that the transfer of files whose sizes are drawn from a heavy-tailed distribution is sufficient to generate self-similarity in network traffic. The ON and OFF periods do not need to have the same distribution. These results suggest that the self-similarity of Web traffic is not a machine-induced artifact; in particular, changes in protocol processing and document display are not likely to remove the self-similarity of Web traffic [Ref. 6].

In a realistic client/server network environment, the degree to which file sizes are heavy-tailed can directly determine the degree of traffic self-similarity at the link level [Ref. 9,10]. This causal relation is proven to be robust with respect to changes in network resources (bottleneck bandwidth and buffer capacity), network topology, the influence of cross-traffic, and the distribution of inter-arrival times. Specifically, measuring self-similarity via the Hurst parameter H and the file size distribution by its power law exponent α, it has been shown that there is a linear relationship between H and α over a wide range of network conditions.

23

The higher the load on the Ethernet, the higher the degree of self-similarity. When the network load is in the range of 30–70 percent, waveforms of the traffic display self-similarity for which H was approximately 1. Furthermore, a load between 80 and 99 percent produces waves with a strong periodic component, and calculation of H becomes unreliable [Ref. 6].

III. DESIGN OF TRAFFIC GENERATOR AGENT

A. CURRENT STATE OF SAAM PROTOTYPE

Two former Naval Postgraduate School students, Dean Vrable and John Yarger, developed

the first working prototype of the proposed SAAM architecture in a Java-based software emulation

environment. The main reason for using Java as the programming language is its portability. Java

programs can run on any platform without changing the precompiled byte code. In addition, Java

has built-in dynamic class loading capability, which provides an ideal platform for evaluating the idea

of extending router services with resident agents. The prototype emulates the routing of IPv6

packets of a SAAM network over a physical IPv4 network environment. Figure 3.1 illustrates the

SAAM prototype.
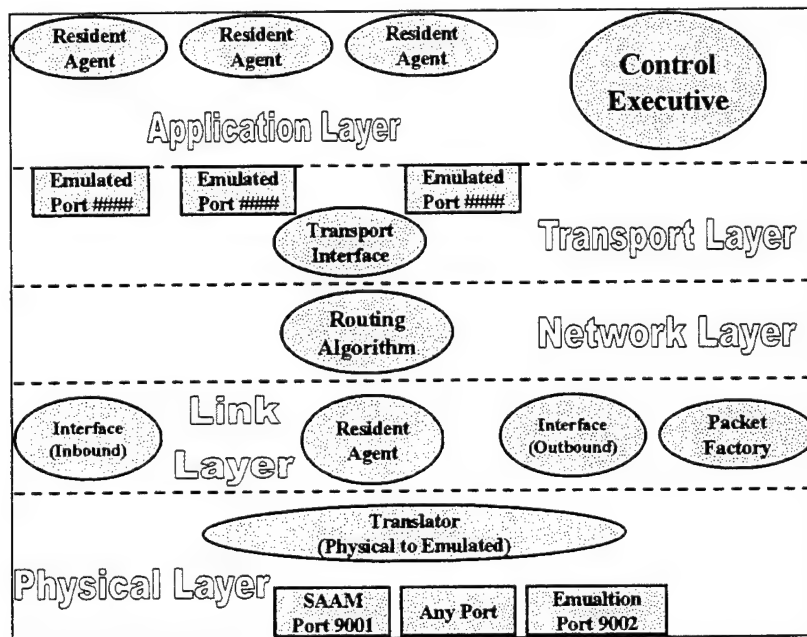
Figure 3.1 SAAM Prototype Protocol Layers

The prototype consists of a router module and a server module. The router is designed to maintain a flow routing table as well as an Address Resolution Protocol (ARP) cache table. The router manages application level flows by using the standard concept of transport layer ports.

The server has two primary tasks. The first task is to maintain an accurate status of its region. This task is accomplished through the building of a Path Information Base (PIB) from auto configuration messages and Link State Advertisement (LSA) messages from the routers. The second task is to respond to flow requests from client applications. This task is accomplished by accessing the PIB to select an optimal path for each flow request. If this optimal path can be found, then the server sends to the routers in the path update for their flow routing tables (FRTE), and notifies the requesting application the assigned *flow Label* for the new flow. Otherwise, the server will reject the flow request and notify the application.

B. REQUIREMENTS OF TRAFFIC GENERATION CAPABILITY

In real networks, only edge routers are connected directly to the clients. Therefore, for the SAAM prototype traffic generators should be installed dynamically on selected routers, instead of residing permanently on each router. Java is perfectly suited for this approach. The SAAM prototype is designed to support dynamic router modules called resident agents, allowing them to be installed, removed, or replaced during runtime. The precompiled byte code of a resident agent is automatically registered as a new module of the target router. After registration, the module may run itself by starting a new thread.

The DemoStation is a logical and artificial entity that is responsible for setting up a SAAM test topology. It creates emulated Primary Server, Backup Server(s) and Routers on computer hosts. Different methods were used for this process. The first one was using a separate *DemoStation*

class file for each new network configuration. In other words, the configuration information is hardcoded. The second one was using ASCII files to store network configuration data. A single *DemoStation* class is used to parse and read these files. The last method was developed and integrated into the current SAAM prototype by an NPS graduate Abanneh [Ref. 5]. This method uses XML documents to store network configuration data. A new class file *saam/demo/SAXParserDemo* was created to parse such a document and read its content. The detailed information about XML and the design of SAAM network configuration specific tags and parser can be found in Abanneh's thesis. His approach enables an external entity (e.g., the DemoStation) to send agents such as *NextNodeProbe* and *PreviousNodeProbe* agents to routers. However, it does not allow a user to specify parameters for an agent. This shortcoming must be addressed for traffic generator agents so that the users may create a wide range of traffic scenarios with ease. The XML document syntax and the *SAXParserDemo* class must be enhanced to allow parameters to be specified for each traffic generator agent instance.

C.   ADDING SUPPORT FOR PASSING PARAMETERS TO AN AGENT

This section explains how the SAAM Prototype has been updated to allow user parameters to be passed to an agent. The DemoStation sends a generic traffic generator agent to a node to initiate the traffic generation process. Before the generation starts, the agent obtains a set of parameters from the DemoStation to create the appropriate traffic flow. This can be achieved by the DemoStation sending the node a new type of message that contains both the byte code for the generic agent and the parameters for the current instance.

To make this approach generic, the message format of agents and the process of retrieving agent parameters from an XML configuration file is designed in such a way that all agents can take advantage of this new capability.

The first step in the design of sending an agent and its parameters together in one message is to develop a new DTD file that works for XML files that contain agent instances with different numbers and types of parameters. Changes to the *SAXParserDemo* class are inevitable so that these new XML files can be parsed and the information gathered can be handled properly. The old class was designed to send agents with no parameters; separate programs had to be used to send parameters to customize these agents. To simplify agent customizations, this class must be modified so that it can send agents with parameters. Also the *DemoInitInfo* class must be modified so that it can store and forward messages in the new agent format.

The second step is to develop a new message format that will carry agents and their parameters to the routers. The message format is shown in Figure 3.2.

Message Type is one byte in length. Routers use this field to differentiate different incoming SAAM control messages. The new agent message format is assigned a unique type value of "30".

Agent Message Format

8	1	variable
Time Stamp	# of Msg	Byte Array of Messages

Agent

1	1	Name length	2	Bytecode length	variable	
Msg Type = 30	Agent Class Name Length	Name of Class File	Agent Bytecode Length	Bytecode	Agent Parameter Strings	... ...

Agent Parameters

1	First Parameter Length	1	Second Parameter Length	
First Parameter String Length	First Parameter String	Second Parameter String Length	Second Parameter String	... ...

Figure 3.2 New Agent Message Format

The first part of the message stores the full class name of an agent. The second part holds the agent's parameters, each of which is represented as a string. The message format supports zero or more (up to 255) parameters. So agents such as *NextNodeProbe* and *PreviousNodeProbe* that have no parameter and agents that have many parameters can be sent using the same message format. The traffic generator agent developed for this thesis take between 2 and 13 input parameters.

The third step is to make changes in the *PacketFactory* and *ControlExecutive* classes so that they can (1) identify this new agent message type, (2) separate agent bytecode from agent parameters, (3) install the agent bytecode, and (4) provide the agent a method to retrieve its parameters.

D.   DESIGN OF FLOWGENERATOR AGENT

A new resident agent named *"FlowGenerator"* is added to the SAAM prototype to implement a generic traffic generator. This agent is designed in such a way that it can be instantiated multiple times in one test topology. Each instance may be customized with a set of input parameters when the instance is created (e.g., inside an XML file). The first parameter decides which traffic model to use to generate traffic. There are four models that can be used. These models are CBR, Poisson, Packet-Train and Self-Similar. The definitions and the usages of these models were explained in Chapter II. Other parameters are traffic model specific or QoS requirement specific.

1.   Generation of CBR Traffic

The inter-packet arrival times are a constant and the constant is obtained using formula (2.6).

$$I \ = \ \frac{1}{R}$$

After creating and sending the first packet, the agent waits for a time duration of I and then generates the next packet. This process continues until the end of the test period. R is a parameter that represents the packet rate in packets per second.

2.   Generation of Poisson Traffic

The inter-packet arrival times are randomly generated using the formula (2.7).

$$I \ = \ \frac{\ln(\ 1 \ - \ U \ )}{- \ R}$$

The amount of wait time between two packet generations, I, follows an exponential distribution with a mean of $1/R$. This random value can be derived based on a pseudo-random generator that chooses a real number U from a uniform distribution in the range of $[0.0, 1.0)$.

3. Generation of Packet Train Traffic

The Packet-Train model has two parameters that directly affect the packet generation process. One of them is R, packet rate. The other parameter is the k, average train size in packets, which is used to figure out inter-train times (Figure 2.1). A Geometric distribution is used to figure out the beginning of an off (inter-train) duration, and an exponential distribution is used to generate inter-train times. Inter-car times (Figure 2.1) are a system parameter. Define

p: the probability that a car is the last of a train (it is used to decide if the car is in this train or in the next).

The train size follows a Geometric distribution, i.e.,

$$\mathbf{Pr}(\textit{Trainsize} = i) = p(1-p)^{i-1} \quad i = 1,2,... \tag{3.1}$$

Therefore,

$$p = 1/k \tag{3.2}$$

After sending each packet, the agent checks if the next packet will be in the same train or not. Specifically, it uses the standard random number generator to produce a real number between [0.0, 1.0]. If this number is smaller than p (the probability that a car is the last of a train), the agent determines that the next packet will be in the next train and the current train ends. Otherwise, the agent creates another packet for the current train and sends it without waiting.

At the end of a train, an inter-train time should be determined. The agent generates this value using an exponential distribution. The mean for the distribution is the reciprocal of the average train rate. The average train size is an input parameter k. The average train rate (denoted by t) can be derived from the average train size and the average packet rate.

t (average train arrival rate) = R (average packet rate) / k (average train size),

31

As an example if $R = 100$ packets per second and $k = 10$ packets, then $t = 100/10 = 10$ trains. The agent generates inter-train times using the same exponential distribution formula (2.7) used for the Poisson model, except that R is replaced with t. At the end of the inter-train time, the agent starts a new train if the test duration is not over yet.

4.   Generation of Self-Similar Traffic

The Self-similar model is used to simulate the aggregated network traffic as explained in Chapter II. A large number of traffic generator instances can be used to create such traffic at run time. But this approach is not the best because it requires the user to specify a huge number of agents and the traffic generators would consumes a lot of CPU, memory, and GUI resources. In the current design, a single traffic generator agent is used for generating self-similar traffic. As soon as the agent is installed, it runs a simulation with multiple ON-OFF traffic sources to generate a sequence of packet generation times and saves this sequence into a temporary trace file. The agent uses this trace file to determine the actual inter-packet arrival times after the actual packet generation process starts.

The simulation part is adapted from a public domain C program developed by Glen Kramer. It is available at "http://wwwcsif.cs.ucdavis.edu/~kramer/trf_gen.html". The Pareto distribution is used to generate the ON and OFF periods for the simulated traffic sources. If the burst size (ON period) is bigger than 0 then a new packet is created. Packet sizes follow a Uniform distribution, ranging between 64 and 1518. The burst size is generated using a Pareto distribution with the parameter "Burst Shape". If the burst size is equal to 0 then an OFF time is generated using a Pareto distribution with the parameter "Inter Burst Shape". A packet generation event consists of two

32

parameters: packet generation time, and packet size in bytes. Packet generation events from different sources are sorted based on their generation times and then written into an ASCII file.

5. FlowSink Agent

In addition to the *FlowGenerator* agent, another agent is designed to receive the packets sent by a *FlowGenerator* agent. This agent is named *FlowSink*. *FlowSink* agent's only function is to receive packets from a specific UDP port and display them.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION OF TRAFFIC GENERATOR AGENT

In order to add traffic generation functionality to the SAAM prototype, some modifications and some additions should have been made to the existing code. In this chapter, these modifications and additions are described.

A. ADDING SUPPORT FOR PASSING PARAMETERS TO AN AGENT

1. Changes to XML DTD File

First, changes are made to the SAAM Configuration DTD file, which is used to validate an XML file that contains a SAAM network configuration. New elements are added to the DTD file to allow an agent to have parameters. These elements are "Agent Name" and "param". All agents must have the "Agent Name" parameter. But some of them, such as *FlowGenerator*, have one or more "param" parameters. The new DTD file and some sample XML files conforming to the new DTD file can be found in Appendix A and B.

2. Changes to SAXParserDemo and Its Inner Classes

The DemoStation uses the *SAXParserDemo* class to load a test topology from an XML file. Some changes are required in the *SAXParserDemo* class and its inner *MyContentHandler* class, which is used to parse XML files. Some new variables are added to the *MyContentHandler* class. These variables are:

- `String tAgentName`
- `String tParam`
- `String [][]agentArray`
- `EmulationPort v4Port`

These variables are used inside the methods of *MyContentHandler* class, such as *startElement(), endElement(), startDocument* and *endDocument()*.

"tAgentName" is used as a temporary storage for the agent's name. When the parser reads an "Agent Name" element in the XML file, the name string is stored in this variable.

35

"tParam" is used as a temporary storage for agent's parameters.

"v4Port" is used to distinguish between nodes that are installed on the same computer. When the parser detects a new node in the XML file, a new v4Port is created for that node. The DemoStation will use this TCP port along with the Ipv4 address of the host computer to communicate with this new node.

"agentArray" is used to store information of all agents defined in an XML file. The size of the array was assigned as [100][20]. This means that it can store 100 different agents each with up to 20 parameters. "tNodeName", "tIPv4","v4Port", and "tAgentName" are common parameters for all agents . The array elements for a particular agent are:

- `agentArray[][0] = tNodeName (Router A, Router B,. .)`
- `agentArray[][1] = tIPV4 (131.120.8.154,. .)`
- `agentArray[][2] = v4Port (9002 ,9004,. .)`
- `agentArray[][3] = tAgentName (FlowGenerator, FlowSink)`
- `agentArray[][4]..agentArray[][19] = Agent's parameters`

Sixteen elements of this array are used to store agent specific parameters. If more than 16 parameters are needed, this limit can be easily changed according to the need. After parsing the XML file, the DemoStation uses the *sendAgent()* method of the *DemoInitInfo* class to send agents to their respective hosting nodes, using the new message format defined in Chapter III.

3. Changes to DemoInitInfo Class

The DemoStation uses this class to deploy the node configuration information to target hosts. This class' methods create routers and servers, activate these nodes by sending them core agents as well as application agents as specified in the XML file. The *sendAgent()* method is modified so that it now can create the new type-'30' SAAM messages that contain both application agents and their parameters. Note that this method can still be used to create packets for agents that have no parameter (with the message type-'0'). Also a new parameter is added to several methods

(sendAgent(), sendPacket(), etc.). This parameter is called DemoStation TCP port number (*v4port*). Different nodes that are to be installed on the same computer are assigned different *v4port* values. (The exact algorithm for assigning these values is implemented in the *Translator* class). For example, if four nodes are installed on the same computer, the first node gets "9002" as its DemoStation Port, second node gets "9004", third gets "9006", and the last one gets "9008". This way, the DemoStation is able to communicate with multiple SAAM nodes on the same computer at the same time via different TCP ports.

The *sendAgent()* method makes calls to the *PacketFactory* class' *append()* and *appendInfo()* methods to build a packet that contains a special message for transporting an agent. "Type of message" for the message is decided by checking if the agent has any parameter or not. If it doesn't have any parameter, the message is assigned type-'0', otherwise it is assigned type-'30'. The DemoStation sends this packet to the target node that is identified by the second and third elements *(tIPV4, v4Port)* of the "agentArray". Specifically, it uses the *sendPacket()* method of the *DemoInitInfo* class.

4. Changes to PacketFactory Class

The *PacketFactory* class is used by the *ControlExecutive* to send and receive SAAM control packets. The *PacketFactory* parses the content of each incoming packet to extract individual SAAM control messages. It then passes these messages to their corresponding message processors. For this task, several blocks of code have been added to the *processPacket()* method for handling of type-'30' (*MESSAGE.TRAFFIC_ GENERATORS*) messages. Specifically, each type-'30' message is broken down into two parts, the agent's byte code and the agent's parameters. The agent's parameters are passed to *ConrolExecutive* by calling the new *setAgentInfo()* method of the

37

ConrolExecutive class. Then the byte code is sent via SAAM_CONTROL_PORT to the *ControlExecutive* class for installation. The *append()* method is also updated to include code for sending traffic agent messages. A new method *appendInfo()* is added. The *DemoInitInfo* class uses this method to add agent parameters to these messages.

5. Changes to ControlExecutive Class

The *ControlExecutive* class maintains control over the event handling mechanism within the SAAM protocol stack by acting as a registrar for agents that wish to communicate via channels or emulated ports.

The *ControlExecutive* class receives an agent from its *receiveEvent()* method. If the agent is an instance of the *ResidentAgent* class, the *replaceOldAgent()* method is called. Inside the *replaceOldAgent()* method, the agent is installed with its *install()* method. Previously, if another instance of the same agent had been installed before on this node, that instance would be uninstalled upon the installation of the new instance. This mechanism is removed so that more than one instance of the *FlowGenerator* and *FlowSink* agents can be installed on the same node. (The agent replacement functionality is left as a future work item.)

Two methods are added to *ConrolExecutive* for *FlowGenerator* and *FlowSink* agents. They are *setAgentInfo()* and *getAgentInfo()*. The *PacketFactory* class uses *setAgentInfo()* to pass agent parameters to *ControlExecutive,* which saves these parameters in an array. Agents use the *getAgentInfo()* method to get their parameters from the *ControlExecutive* class.

B. TRAFFIC GENERATOR AGENT IMPLEMENTATION

FlowGenerator and *FlowSink* agents are implemented into the SAAM test bed by adding some new classes.

38

1. FlowGenerator Class

A Java source file named *"FlowGenerator.java"* is added to the *saam.agent.applications* package, which includes all the application level resident agents that can be deployed to a router. The new file contains the code for the *FlowGenerator* class, which implements the *ResidentAgent* and *Runnable* interfaces.

Implementing *ResidentAgent* is a necessary requirement for a *FlowGenerator* object to be a resident agent. The registration and deregistration of resident agents are handled by the *ControlExecutive* class.

Implementing *Runnable* allows a *FlowGenerator* object to be executed by a thread. Interface *Runnable* is designed to provide a common protocol for objects that wish to execute as a separate thread in the JVM. The class structure of *FlowGenerator* agent is shown in Figure 4.1.

ControlExecutive installs an agent by calling the agent's *install()* method. Inside of the *install()* method, the agent may obtain its parameters by calling the *getAgentInfo()* method of the *ControlExecutive* class. Every time a new *FlowGenerator* agent instance is installed, a new thread is created to handle the jobs of this new agent.

The *run()* method of the class is automatically called when the thread is started inside the *install()* method. The first thing that the agent does is to obtain its parameters from *ControlExecutive*. The list of parameters may be different from instance to instance, depending on the traffic model and the QoS type specified.
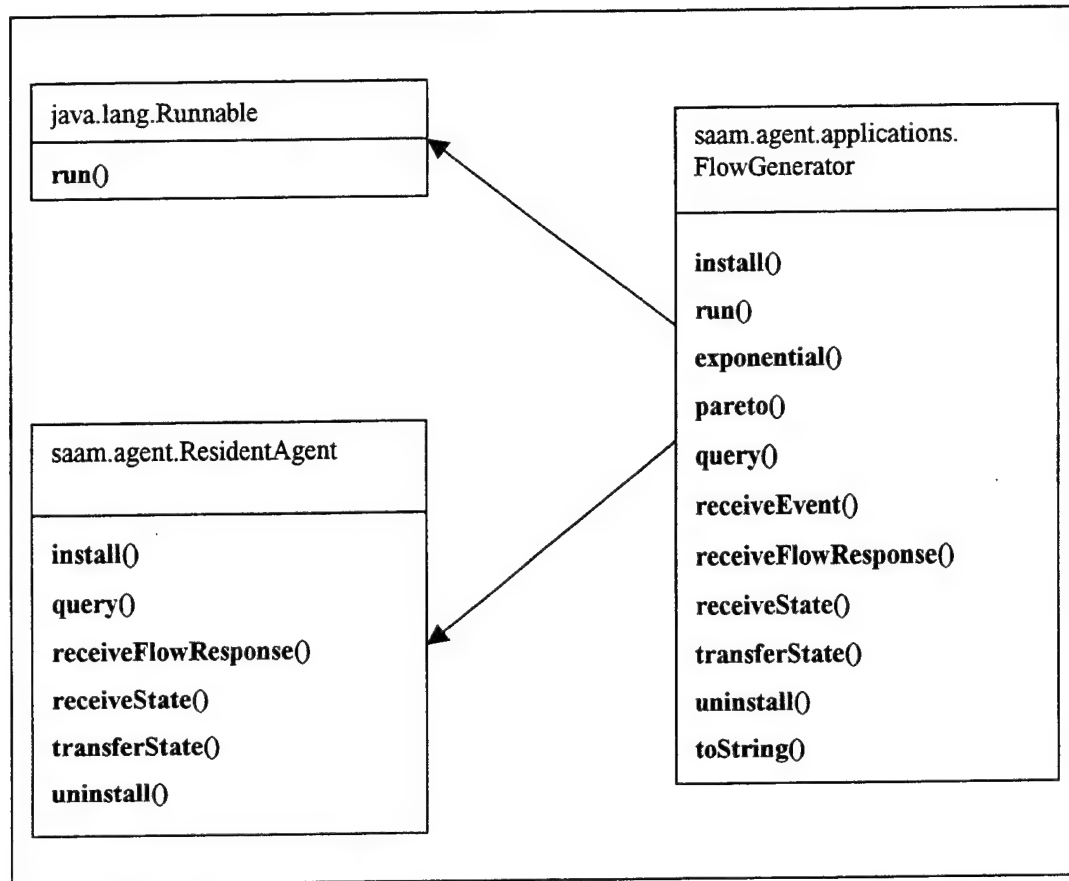
Figure 4.1 FlowGenerator Class Structure

For each traffic generator agent instance defined in an XML configuration file, one of these four traffic models may be specified: CBR, Poisson, Packet-Train, Self-Similar models. Each traffic model requires a different number of parameters. One of these three QoS types may also be specified for the agent: Guaranteed QoS (IntServ), Differentiated QoS (DiffServ), and Best-Effort. Each QoS type requires a different number of parameters. The parameters required by different traffic models and different types of QoS requests are shown in Figure 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7.

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterA-Poisson-1</Param>
    <!-- Instance Name-->
    <Param>99.99.99.99.18.0.0.0.0.0.0.0.0.0.0.1</Param>
    <!-- Destination Ipv6 address-->
    <Param>6000</Param>
    <!-- Destination Port number (int)-->
    <Param>IntServ</Param>
    <!-- Type of Service-->
    <Param>0.02</Param>
    <!-- Requested Loss Rate ( % Percentage)-->
    <Param>15</Param>
    <!--Requested Delay (Millisecond)-->
    <Param>400</Param>
    <!-- initial wait  (Millisecond)-->
    <Param>200</Param>
    <!-- Test Duration (Millisecond)-->
    <Param>Poisson</Param>
    <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar)-->
    <Param>325</Param>
    <!-- Packet rate per second-->
    <Param>1415</Param>
    <!-- Payload Size-->
</Agent>
```

Figure 4.2 Parameters of FlowGenerator Agent (Poisson, IntServ)

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterB-Poisson-1</Param>
    <!-- Instance Name-->
    <Param>99.99.99.99.23.0.0.0.0.0.0.0.0.0.0.1</Param>
    <!-- Destination Ipv6 address-->
    <Param>6000</Param>
    <!-- Destination Port number (int)-->
    <Param>DiffServ</Param>
    <!-- Type of Service (IntServ, DiffServ, Best-Effort)-->
    <Param>1</Param>
    <!--User ID (1, 2 , 3, 4, 5)-->
    <Param>400</Param>
    <!-- initial wait (Millisecond)-->
    <Param>200</Param>
    <!-- Test Duration (Millisecond)-->
    <Param>Poisson</Param>
    <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar)-->
    <Param>350</Param>
    <!-- Packet rate per second-->
    <Param>1415</Param>
    <!-- Payload Size-->
</Agent>
```

Figure 4.3 Parameters of FlowGenerator Agent (Poisson, DiffServ)

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterC-CBR-1</Param>
    <!--Instance Name-->
    <Param>99.99.99.99.23.0.0.0.0.0.0.0.0.0.0.2</Param>
    <!-- Destination Ipv6 address-->
    <Param>6001</Param>
    <!-- Destination Port number-->
    <Param>DiffServ</Param>
    <!-- Type of Service (IntServ, DiffServ, Best-Effort)-->
    <Param>2</Param>
    <!--User ID (1, 2 , 3, 4, 5)-->
    <Param>400</Param>
    <!-- initial wait -->
    <Param>240</Param>
    <!-- Test Duration-->
    <Param>CBR</Param>
    <!-- Distribution-->
    <Param>225</Param>
    <!-- Packet rate per second-->
    <Param>1320</Param>
    <!-- Payload Size-->
</Agent>
```

Figure 4.4 Parameters of FlowGenerator Agent (CBR, DiffServ)

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterC-PacketTrain-1</Param>
    <!-- Instance Name-->
    <Param>99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2</Param>
    <!-- Destination Ipv6 address-->
    <Param>6001</Param>
    <!-- Destination Port number-->
    <Param>DiffServ</Param>
    <!-- Type of Service (IntServ, DiffServ, Best-Effort)-->
    <Param>2</Param>
    <!--User ID (1, 2 , 3, 4, 5)-->
    <Param>400</Param>
    <!-- initial wait -->
    <Param>210</Param>
    <!-- Test Duration-->
    <Param>Packet-Train</Param>
    <!-- Distribution-->
    <Param>300</Param>
    <!-- Packet rate per second-->
    <Param>999</Param>
    <!-- Payload Size-->
    <Param>8</Param>
    <!-- Average Train Size-->
</Agent>
```

Figure 4.5 Parameters of FlowGenerator Agent (Packet-Train, DiffServ)

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterE-SelfSimilar-1</Param>
    <!--Instance Name-->
    <Param>99.99.99.99.17.0.0.0.0.0.0.0.0.0.2</Param>
    <!-- Destination Ipv6 address-->
    <Param>6000</Param>
    <!-- Destination Port number  (int)-->
    <Param>IntServ</Param>
    <!-- Type of Service (IntServ, DiffServ, Best-Effort)-->
    <Param>0.02</Param>
    <!-- Requested Loss Rate ( % Percentage)-->
    <Param>12</Param>
    <!--Requested Delay (Milisecond)-->
    <Param>400</Param>
    <!-- initial wait  (Milisecond)-->
    <Param>200</Param>
    <!-- Test Duration  (Milisecond)-->
    <Param>Self-Similar</Param>
    <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar)-->
    <Param>150000</Param>
    <!--Bandwidth (Kbps)-->
    <Param>0.19</Param>
    <!--load (float)-->
    <Param>10</Param>
    <!--number of sources (int)-->
    <Param>10000</Param>
    <!--number of packets (int)-->
</Agent>
```

Figure 4.6 Parameters of FlowGenerator Agent (Self-Similar, IntServ)

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterJ-SelfSimilar-1</Param>
    <!--Instance Name-->
    <Param>99.99.99.99.21.0.0.0.0.0.0.0.0.0.1</Param>
    <!-- Destination Ipv6 address-->
    <Param>6000</Param>
    <!-- Destination Port number  (int)-->
    <Param>Best-Effort</Param>
    <!-- Type of Service (IntServ, DiffServ, Best-Effort)-->
    <Param>400</Param>
    <!-- initial wait  (Milisecond)-->
    <Param>200</Param>
    <!-- Test Duration  (Milisecond)-->
    <Param>Self-Similar</Param>
    <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar)-->
    <Param>200000</Param>
    <!--Bandwidth (Kbps)-->
    <Param>0.25</Param>
    <!--load (float)-->
    <Param>9</Param>
    <!--number of sources (int)-->
    <Param>10000</Param>
    <!--number of packets (int)-->
</Agent>
```

Figure 4.7 Parameters of FlowGenerator Agent (Self-Similar, Best-Effort)

43

After a traffic generator agent retrieves its parameters, it must wait for a period of time before generating traffic. This time can be specified using the "initialDelay" parameter. A minimum wait time is required for the server to build the network topology in its database. Also with this parameter, all traffic generators can be deployed at once even though they may start at different times in the test scenario. Before going into a wait mode, a self-similar traffic generator also calls the constructor of the *TraceGenerator* class with the following parameters; "instanceName", "bandwidth", "load", "sources". *TraceGenerator* creates instances of *SingleSource* class (ON-OFF traffic sources), which then generates packets to form a Self-Similar traffic trace. Details on *TraceGenerator* will be presented later.

At the end of the "initialDelay", the traffic generator sends a *FlowRequest* message to the server and waits for a *FlowResponse* message except that it is specified to use Best-Effort QoS. A Best-Effort traffic generator starts generating packets immediately without sending a *FlowRequest* message. Other types of traffic generators start generating traffic only after they have received a positive response to their flow requests.

2.   TraceGenerator Class

A Java source file named *"TraceGenerator.java"* is added to the *saam.agent.applications* package. Self-Similar *FlowGenerator* agents use this class to generate traffic that has Self-Similar characteristics. The new file contains the code for the *TraceGenerator* class and its inner *SingleSource* class. Four parameters are passed from the *FlowGenerator* class when *TraceGenerator* class' constructor is called. The first one is "instanceName", which is used to name the trace file created so that trace files created by different Self-Similar agent instances can be distinguished. (The trace files are saved in the following directory: agent\applications\temp.)

The others are "bandwidth", "load", and "sources", which are used to generate traffic that has a bit rate equal to the product of "load" and "bandwidth" using "sources" number of ON-OFF sources. Another call is made to the *outputTraces()* method of the *TraceGenerator* class from class *FlowGenerator* to pass the parameter "testDuration". This parameter is used inside the *extractPacket()* method of the *SingleSource* class to stop the packet trace creation process. The class structures of *TraceGenerator* and *SingleSource* are shown in Figure 4.8.

saam.agent.applications. TraceGenerator	saam.agent.applications. SingleSource
TraceGenerator() **addSource()** **addSources()** **generateTrace()** **getBytes()** **getLoad()** **getPackets()** **getTime()** **outputTraces()** **sort()** **uniform()**	**SingleSource()** **exponent()** **extractPacket()** **getArrival()** **getPacketSize()** **pareto()** **randomValue()** **uniform()** **uniformNon0()** **reset()**
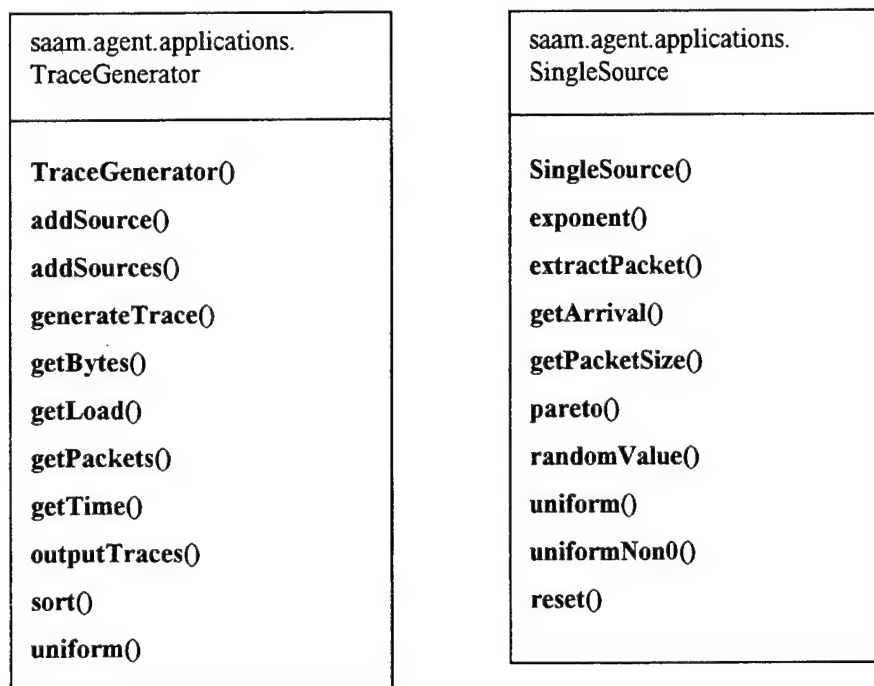
Figure 4.8 TraceGenerator and SingleSource Class Structures

3.   FlowSink Class File

A Java source file named *"FlowSink.java"* is added to the *saam.agent.applications* package. The *FlowSink* resident agent implements the *ResidentAgent*, and *Runnable* interfaces.

Components in the SAAM prototype can listen to particular *channels* to receive event notification by implementing the *SaamListener* interface. *Channels* are objects that serve as glue

between components, passing data between them via events. Components registered to listen to a channel will receive an instance of each event sent to the channel. When an agent wants to listen to a particular channel, the agent has to register the intent by using the *addListenerToChannel()* or *monitorPort()* methods in the *ControlExecutive* class .

All resident agents automatically implement the *SaamListener* interface by implementing the *ResidentAgent* interface. A *FlowSink* agent has to monitor a particular network interface to receive a copy of every packet coming through that interface. The *FlowSink* class structure is shown in Figure 4.9.

A *FlowSink* agent has two parameters: "instanceName" and "portNumber". "portNumber" determines where the agent will listen for incoming packets. *FlowSink* agents act as addressable UDP sinks for packets generated by *FlowGenerator* agents. They do not generate packets or send acknowledgements. Like *FlowGenerator*, each instance of *FlowSink* is designed to run on a separate thread. Multiple *FlowSink* agents can be installed on the same router with different parameters.
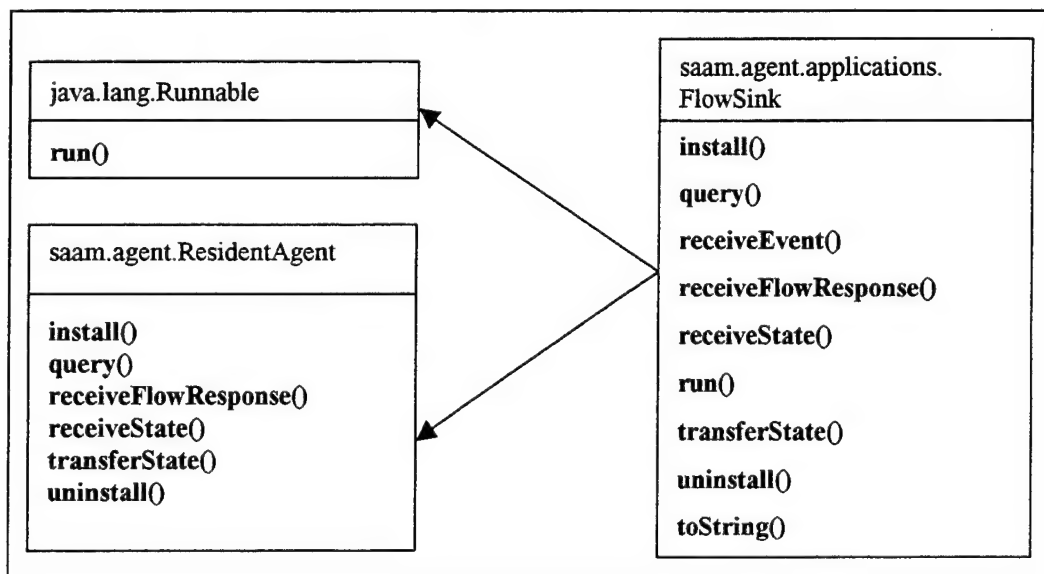
Figure 4.9 FlowSink Class Structure

A sample *FlowSink* agent parameter list is shown in Figure 4.10.

```
<Agent>
    <AgentName>FlowSink</AgentName>
    <Param>RouterA-FlowSink-1</Param>
    <!--Instance Name-->
    <Param>6000</Param>
    <!-- receiver port number (int)-->
</Agent>
```

Figure 4.10 Parameters of FlowSink Agent

THIS PAGE INTENTIONALLY LEFT BLANK

V. TESTS AND RESULTS

The goal of these tests is to see that the new Traffic Generator agent is functioning as planned. After the integration of the traffic agent code into the SAAM prototype, the new traffic generation capability is evaluated on different test topologies that contain a variety of instances of the agent. The results are reported in this chapter.

A. TEST I

The system was tested first with the network topology shown in Figure 5.1. Although the topology is simple, it allows a quick test of the basic functionalities of the Traffic Generator. This topology is stored in the XML file: "demo/1Server-3Router-1Host.xml".
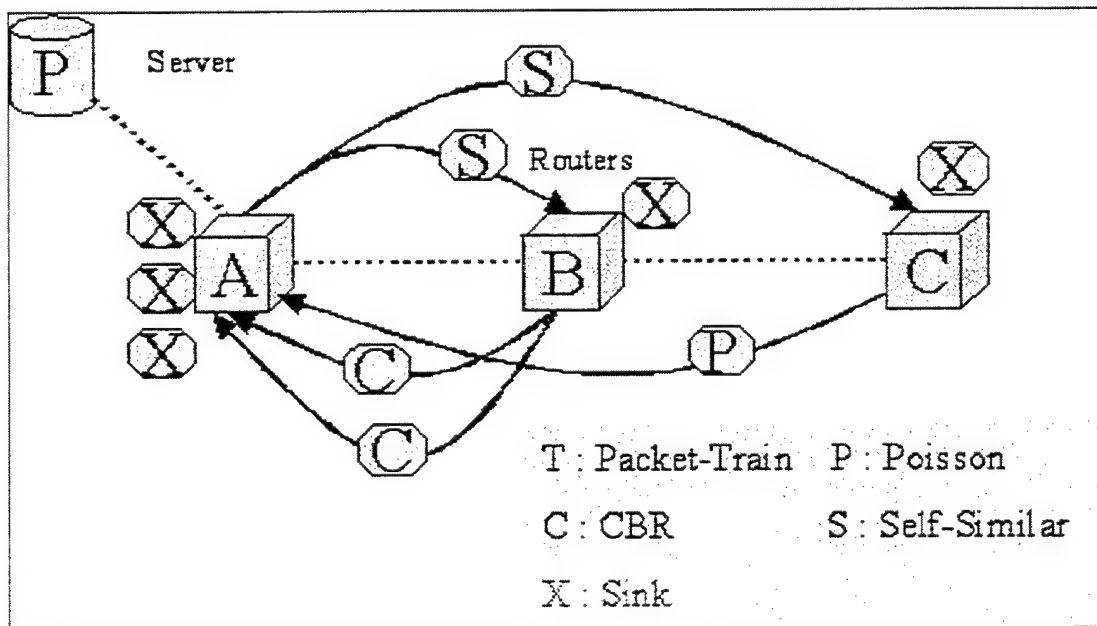
Figure 5.1 Trafic Agent Test Topology I

In this topology, there is one primary server and three routers. Five traffic agents are installed on Router A, two *FlowGenerator* and three *FlowSink* agents. Both of the *FlowGenerator* agents generate Self-Similar traffic. The *FlowSink* agents, which are installed on Router B and

Router C respectively, receive the packets coming from the generator agents. One of the generator agents (RouterA-SelfSimilar-1) requests Integrated Service; the other (RouterA-SelfSimilar-2) requests Differentiated Service.

Router B has two *FlowGenerator* agents and one *FlowSink* agent. Both of the *FlowGenerator* agents generate CBR traffic that is destined for the *FlowSink* agents installed on Router A. One of the agents (RouterB-CBR-1) requests Integrated Service while the other (RouterB-CBR-2) requests Differentiated Service.

Router C has one *FlowGenerator* agent that generates Poisson traffic that goes to the *FlowSink* agent installed on Router A. This agent uses Best-Effort Service to send its packets. The arrows on the Figure 5.1 show the traffic flow directions. Figure 5.2, 5.3, and 5.4 show the agents installed on routers.

Figure 5.2 Snapshot of Router A Traffic Agents

Figure 5.3 Snapshot of Router B Traffic Agents
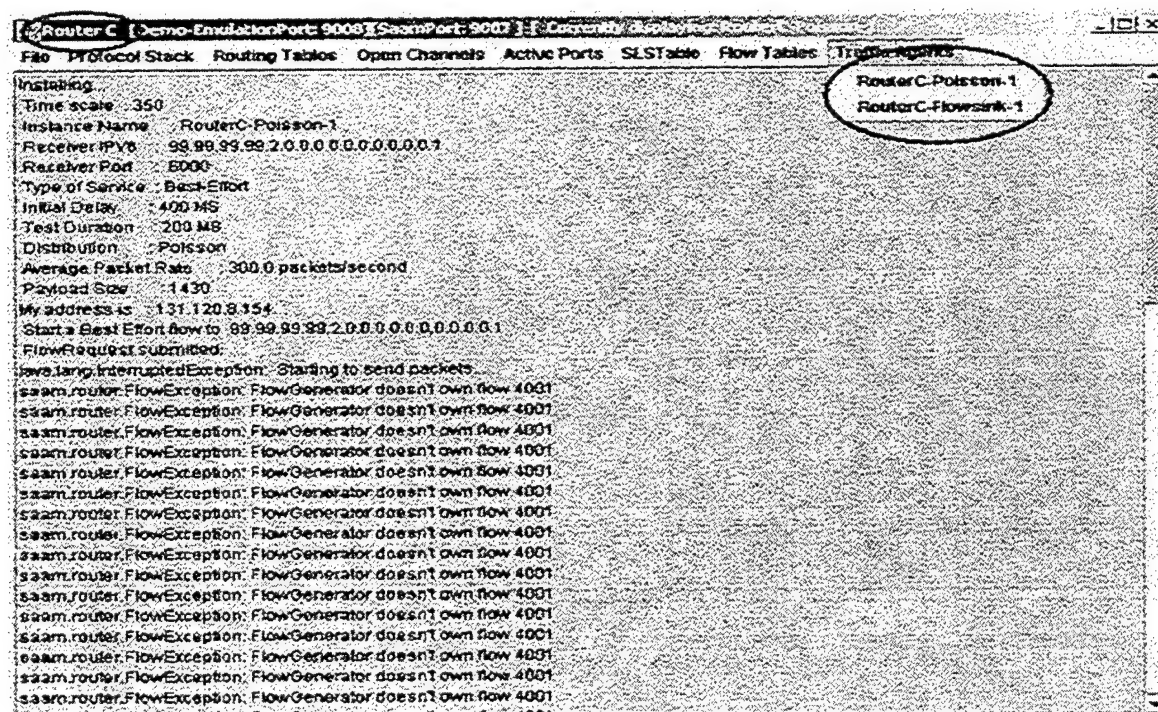
Figure 5.4 Snapshot of Router C Traffic Agents

51

The parameters used by the traffic generator agents in this test are shown in Figure 5.5, 5.6, 5.7, and 5.8.

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterA-SelfSimilar-1</Param>            <!--Instance Name-->
    <Param>99.99.99.99.3.0.0.0.0.0.0.0.0.0.1</Param>   <!-- Destination Ipv6 address-->
    <Param>6000</Param>                             <!-- Destination Port number (int)-->
    <Param>IntServ</Param>                          <!-- Type of Service (IntServ, DiffServ, Best-Effort)-->
    <Param>0.01</Param>                             <!-- Requested Loss Rate (% Percentage)-->
    <Param>10</Param>                               <!--Requested Delay (Milisecond)-->
    <Param>400</Param>                              <!-- Initial wait (Milisecond)-->
    <Param>200</Param>                              <!-- Test Duration (Milisecond)-->
    <Param>Self-Similar</Param>                     <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar)-->
    <Param>100000</Param>                           <!--Bandwidth (Kbps)-->
    <Param>0.25</Param>                             <!--Load (float)-->
    <Param>9</Param>                                <!--number of sources (int)-->
    <Param>10000</Param>                            <!--number of packets (int)-->
</Agent>
```
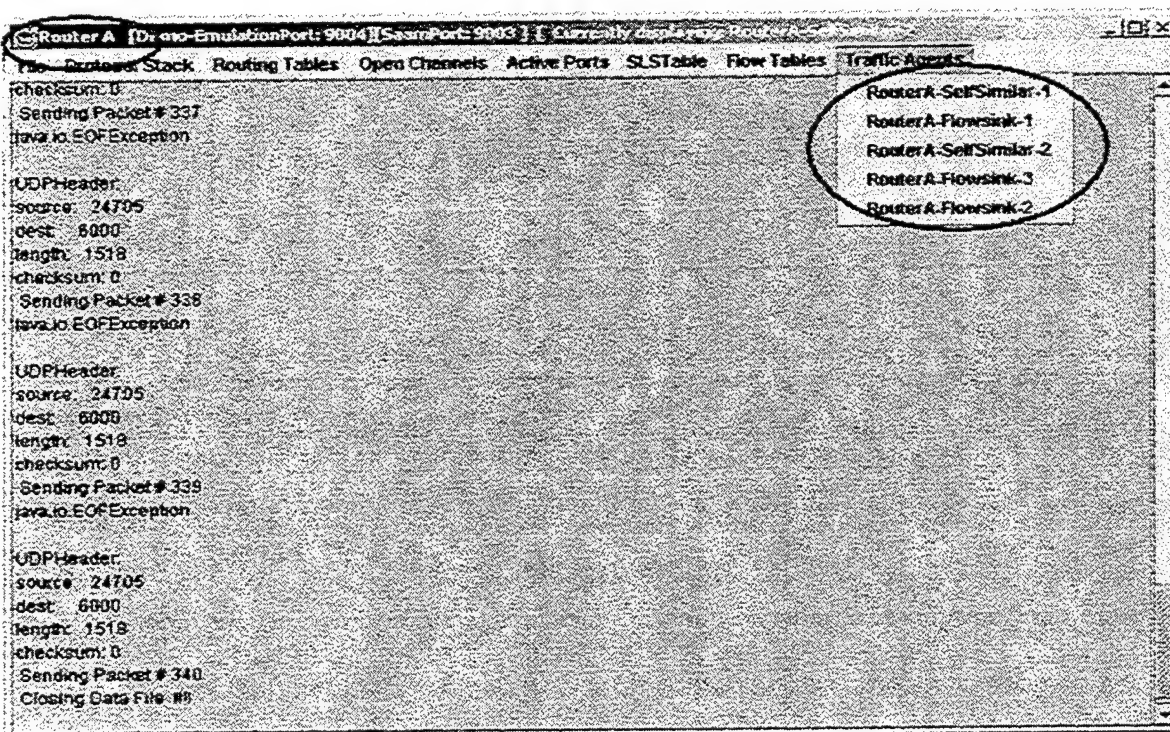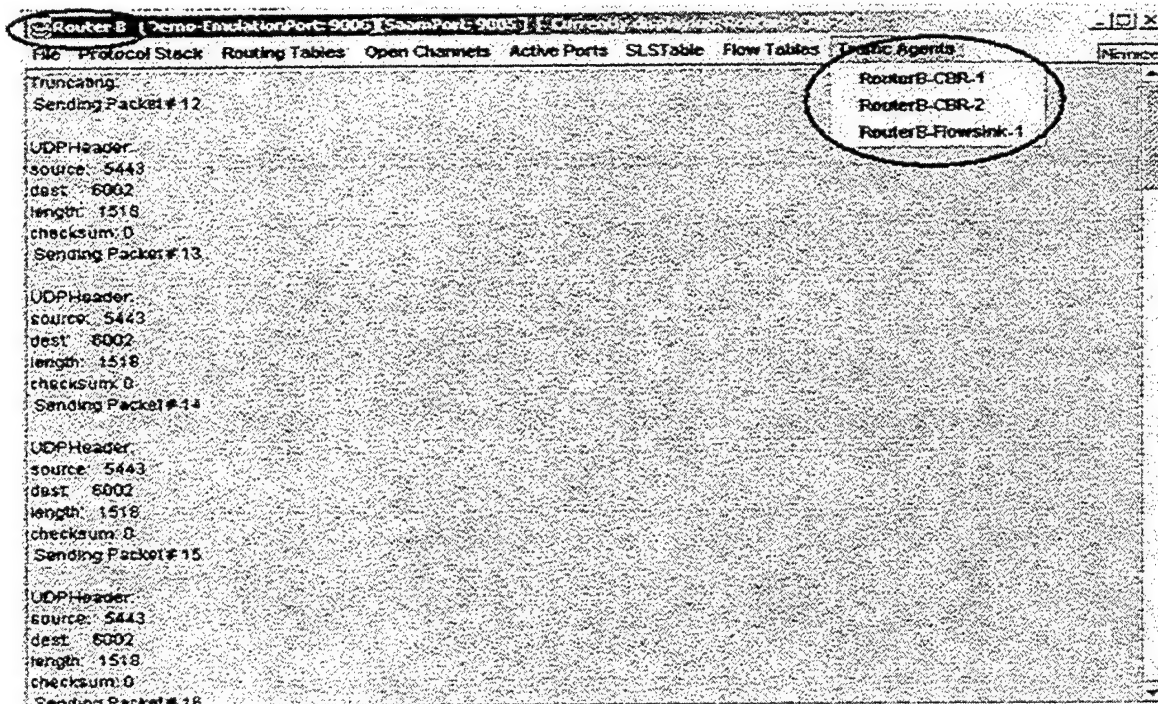
Figure 5.5 Parameters of Agent RouterA-SelSimilar-1

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterA-SelfSimilar-2</Param>            <!--Instance Name-->
    <Param>99.99.99.99.3.0.0.0.0.0.0.0.0.0.2</Param>   <!-- Destination Ipv6 address-->
    <Param>6000</Param>                             <!-- Destination Port number (int)-->
    <Param>DiffServ</Param>                         <!-- Type of Service (IntServ, DiffServ, Best-Effort)-->
    <Param>1</Param>                                <!-- USer_ID (1,2,3,4,5)-->
    <Param>400</Param>                              <!-- Initial wait (Milisecond)-->
    <Param>200</Param>                              <!-- Test Duration (Milisecond)-->
    <Param>Self-Similar</Param>                     <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar)-->
    <Param>100000</Param>                           <!--Bandwidth (Kbps)-->
    <Param>0.24</Param>                             <!--Load (float)-->
    <Param>11</Param>                               <!--number of sources (int)-->
    <Param>10000</Param>                            <!--number of packets (int)-->
</Agent>
```

Figure 5.6 Parameters of Agent RouterA-SelSimilar-2

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterB-CBR-1</Param>                        <!-- Instance Name -->
    <Param>99.99.99.99.2.0.0.0.0.0.0.0.0.0.1</Param>    <!-- Destination Ipv6 address -->
    <Param>6001</Param>                                 <!-- Destination Port number (int) -->
    <Param>IntServ</Param>                              <!-- Type of Service (IntServ, DiffServ, Best-Effort) -->
    <Param>0.01</Param>                                 <!-- Requested Loss Rate ( % Percentage) -->
    <Param>10</Param>                                   <!-- Requested Delay (Milisecond) -->
    <Param>400</Param>                                  <!-- Initial wait (Milisecond) -->
    <Param>200</Param>                                  <!-- Test Duration (Milisecond) -->
    <Param>CBR</Param>                                  <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar) -->
    <Param>200</Param>                                  <!-- Packet Rate per Second (Float) -->
    <Param>1410</Param>                                 <!-- Payload Size (byte) -->
</Agent>
```

Figure 5.7 Parameters of Agent RouterB-CBR-1

```
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>RouterC-Poisson-1</Param>                    <!-- Instance Name -->
    <Param>99.99.99.99.2.0.0.0.0.0.0.0.0.0.1</Param>    <!-- Destination Ipv6 address -->
    <Param>6000</Param>                                 <!-- Destination Port number (int) -->
    <Param>Best-Effort</Param>                          <!-- Type of Service (IntServ, DiffServ, Best-Effort) -->
    <Param>400</Param>                                  <!-- Initial wait (Milisecond) -->
    <Param>200</Param>                                  <!-- Test Duration (Milisecond) -->
    <Param>Poisson</Param>                              <!-- Distribution (CBR, Poisson, Packet-Train, Self-Similar) -->
    <Param>300</Param>                                  <!-- Packet Rate per Second (Float) -->
    <Param>1430</Param>                                 <!-- Payload Size (byte) -->
</Agent>
```

Figure 5.8 Parameters of Agent RouterC-Poisson-1

FlowSink agents only receive "instance Name" and "port number" as a parameters. A sample

snapshot of a *FlowSink* agent is shown in Figure 5.9 and its parameters in Figure 5.10.

53
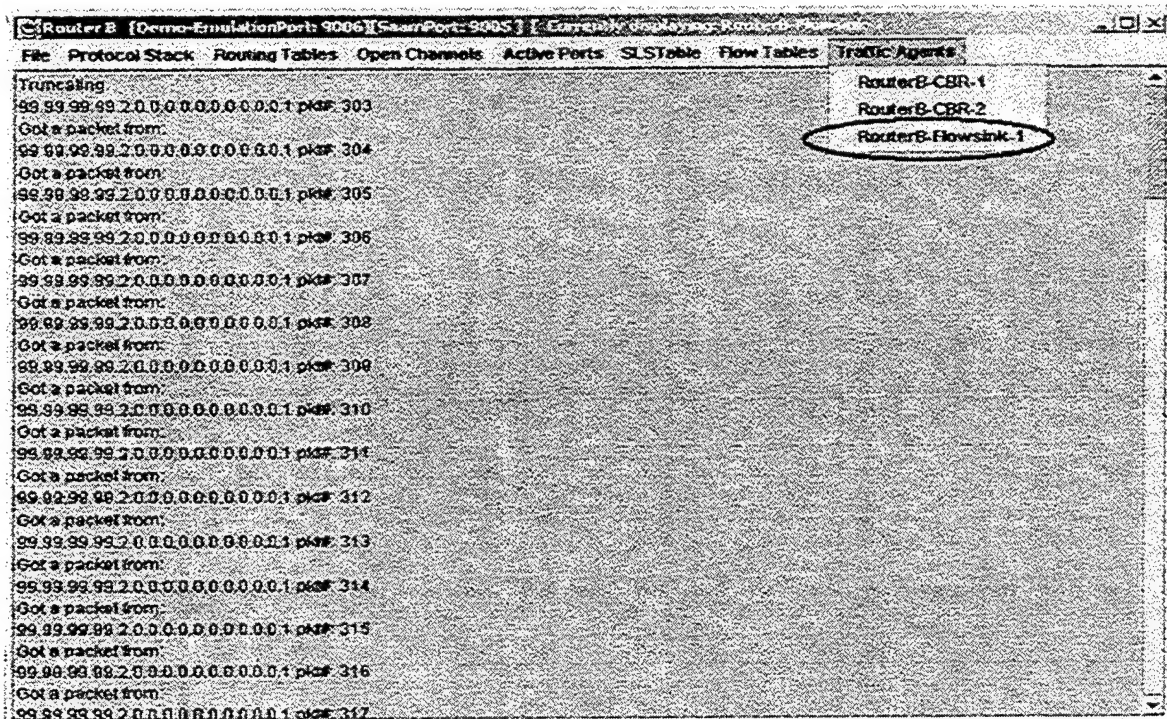
Figure 5.9 Snapshot of Agent RouterB-FlowSink-1
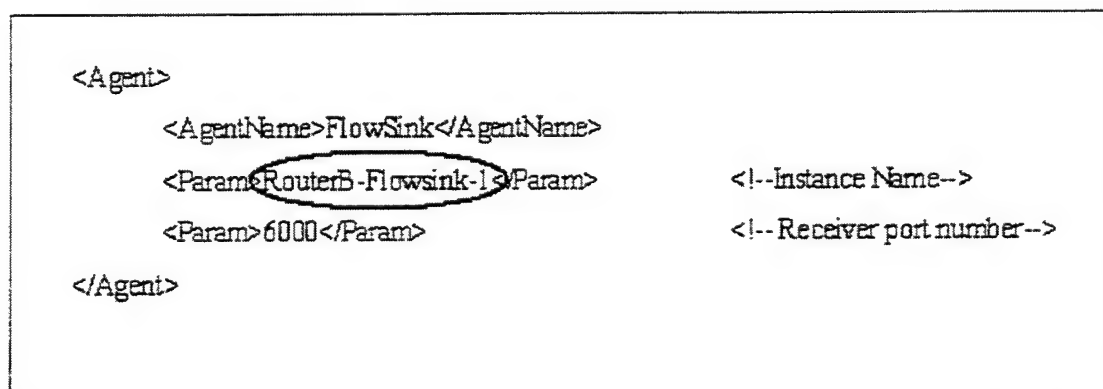
Figure 5.10 Parameters of Agent RouterB-Flowsink-1

The topology shown in Figure 5.1 was deployed three times and the behaviors of the traffic agents were observed. The tests were successful. All agents were installed correctly and they obtained their parameters and generated packets as expected. All packets created by the generator agents were delivered to their respective sinks.

54

B.   TEST II

The system was then tested with a network topology that is a little bit more complex than the first one. The topology is shown in Figure 5.11. This topology is stored in the XML file: "demo/1Server-7Router-2Host.xml".
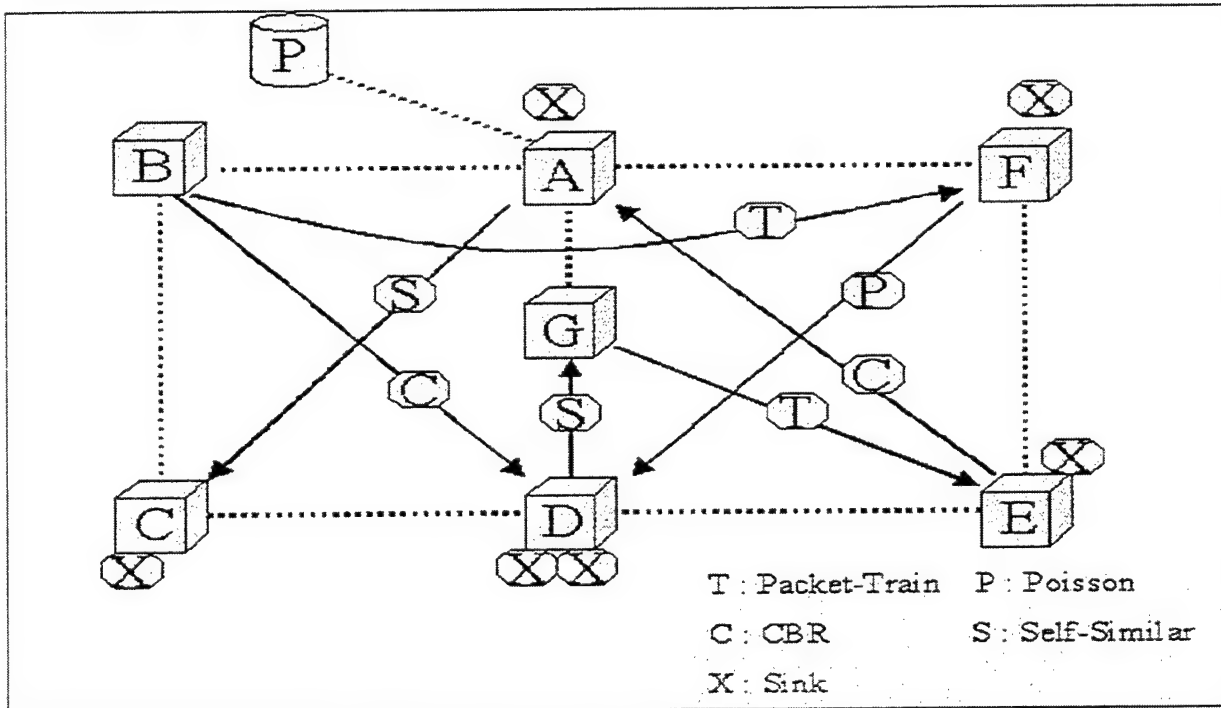
Figure 5.11 Trafic Agent Test Topology II

Seven routers are used in this test topology, with many *FlowGenerator* and *FlowSink* agents installed on them. Router A has one *FlowGenerator* and one *FlowSink*. The *FlowGenerator* agent sends Self-Similar traffic to Router C. Router B has two *FlowGenerator* agents. One of them sends Packet-Train traffic to Router F. The other sends CBR traffic to Router D. Router C only has one *FlowSink* agent. Router D has two *FlowSink* agents and one *FlowGenerator* agent sending Self-Similar traffic to Router G. Router E has one *FlowSink* agent and one *FlowGenerator* agent that sends CBR traffic to Router A. Router F has one *FlowSink* agent and one *FlowGenerator* agent that

sends Poisson traffic to Router D. Router G also has one *FlowSink* agent and one *FlowGenerator* agent that sends Packet-Train traffic to Router E.

After a couple of deployments of this test topology, everything seemed to work fine except that two problems were observed. One of them was that some routers discarded a lot of packets coming to or passing through them. This problem might be caused by a bug in the "Routing Algorithm" module in SAAM, which only would show up in the presence of heavy traffic volume. The second problem was that DiffServ and Best Effort flows did not reach their destinations. This is because routing of DiffServ or Best Effort packets have not been fully implemented in the SAAM prototype. However, the traffic generator agents received their parameters from the DemoStation correctly. A snapshot of the GUI for one of these agents, Router-C-Poisson-1, is shown in Figure 5.4.

Another concern was the Self-Similar model itself. It was hard to determine if a self-traffic generator really generated Self-Similar traffic with the load specified by the user. The trace files, which were created by Self-Similar agents, were plotted out using Microsoft Excel. One of the traces, created by the Self-Similar generator at Router A, is shown in 5.12. The parameters of the agent can be seen in Figure 5.5.
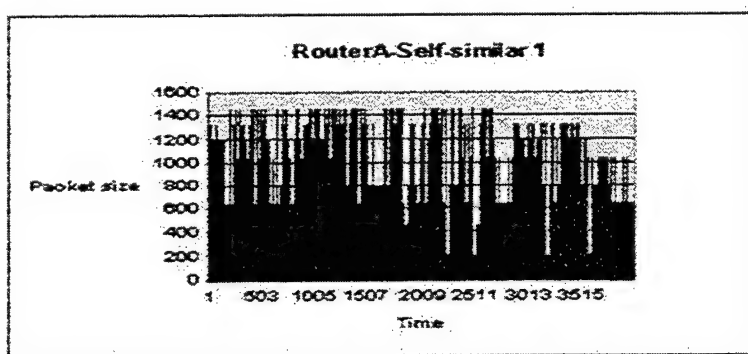
Figure 5.12 Plot of Self-Similar Traffic Trace Generated at Router A

56

Figure 5.13 shows how the server constructs its PIB during a test. Figure 5.14 shows how the server handles IntServ flow requests sent by the traffic generator agents.

Figure 5.13 Snapshot of Server PIB GUI.

Figure 5.14 Snapshot of Server PIB GUI.

C.   TEST III

The planned last test was to deploy the topology shown in Figure 5.15. This network topology is similar to the Naval Postgraduate School's network topology. Fourteen routers were selected and one Primary and one Backup server were added to the topology. Many *FlowGenerator* and *FlowSink* agents were installed on each router. Each traffic agent is identified by a small circle with a letter inside indicating the traffic agent type. This topology is stored in the XML file: "demo/NPS-4Host.xml".

The results of this test have not been fully collected due to the same problems as in Test II. It was observed however, that all routers, servers and agents were installed correctly with the specified parameters. Four computers were used to run this topology.

The connections between Routers A-B-C-D-E-F-G and Primary Server in Figure 5.15 actually form the same topology as the one used in test II.
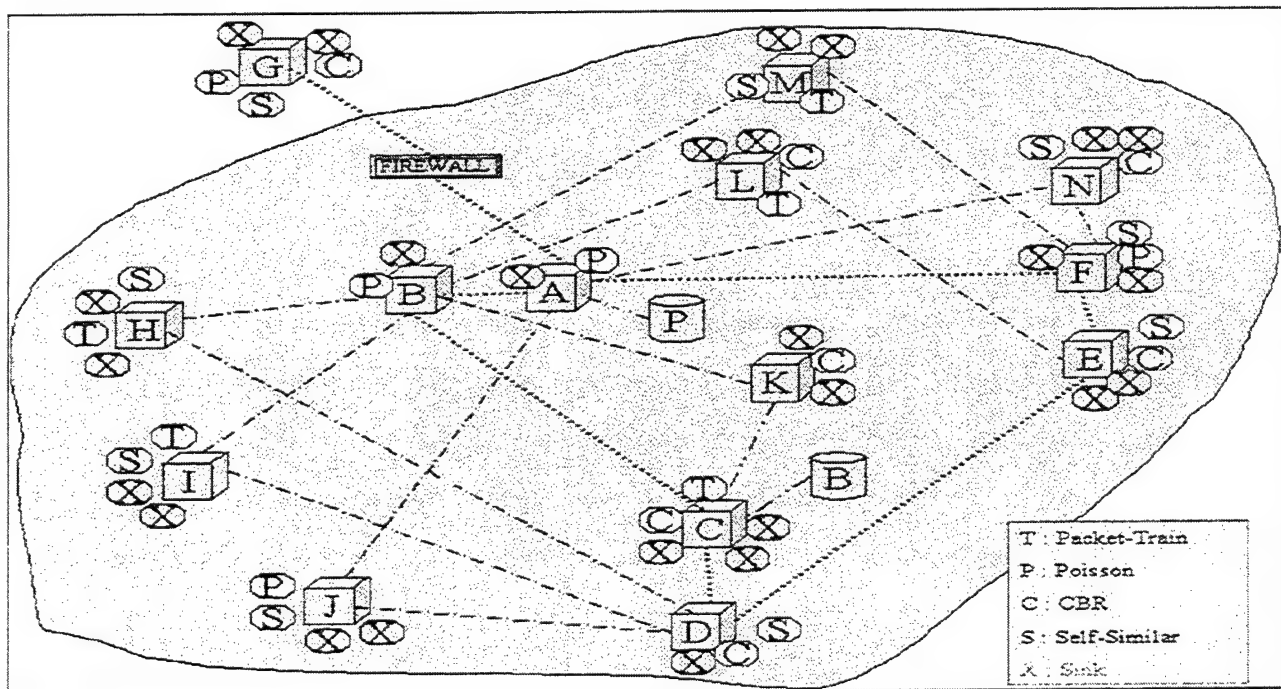
Figure 5.15 Trafic Agent Test Topology III

VI. CONCLUSION

A.   LESSONS LEARNED

A general-purpose user traffic generation capability has been developed for the SAAM prototype. The implementation has been tested after its integration into the current SAAM prototype. The work also provides a simple but robust mechanism to customize the behavior of a resident agent by allowing parameters to be specified for the agent in the XML configuration file.

1.   Integration

The code size of the SAAM prototype implementation is getting bigger and bigger, and one of the more difficult parts of the thesis was the integration of traffic generator agent code into the current SAAM prototype without affecting other working components of the System. Some unexpected code errors occurred, which were very difficult to debug without first acquiring a thorough knowledge of the existing system.

B.   FUTURE WORK

1.   Replacement of Existing Agent

The ability to replace an existing agent has been deactivated so that more than one traffic generators or sinks can be installed on the same node. The old code allowed only one instance of any agent on one node. When the new agent was installed, the old one with the same class name was uninstalled. There are two issues that should be looked at closely when designing a new agent replacement capability:

- The new capability should be backward compatible, i.e., it will allow more than one instances of an agent on the same node, and at the same time it will allow the replacement of an existing instance.

- It should make it possible and simple to selectively replace one of the many existing instances of an agent.

2. Validation of Self-Similar Model

The Self-Similar trace generation method used by the *FlowGenerator* class has not been fully validated. The packet generation information is currently available in ASCII-based trace files (in the "agent/applications/temp" directory). The correctness of the Self-Similar model can be validated by examining these files using the methods described in Chapter II. These methods are:

- Analysis of the variances of the aggregated processes $X^{(m)}$.

- Analysis of the rescaled range (R/S) statistic for different block sizes.

- Use of Whittle estimator.

APPENDIX A. SAAM CONFIGURATION DTD FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) -->
<!--DTD generated by XML Spy v3.0.7 NT (http://www.xmlspy.com)-->
<!-- The DTD file enforces the rules under which the XML file is composed. It defines the structure
of the XML file.
 If the XML file breaks any rule then that file will be invalid.
XMLSPY validates the file and decides if it conforms to the rules, but the user has the choice to
proceed without validity. In that case unacceptable results can happen. In SAAM configuration
XML file a non-valid file will cause a parsing error and the configuration information will not be
extracted from the file. -->
<! -- The rules enforced by the DTD are:
1· Each node in SAAM requires some elements to be mandatory. NodeType, NodeName, IPv4 and
TimeScale  are mandatory  and there should be only one of these elements for servers and routers.
2· Some of them can be included in the Server node, but not the Router node. SC_GlobalWaitTime,
SC_LocalWaitTime,  SC_CycleTime and SC_MetricType are exclusively for the servers, which are
either Primary or Backup. Only one value of each element should be included for a Server node.
These values are required to instantiate a server, but routers do not require them. Thus, a node can
have zero or one of these elements.
3· All nodes are required to have at least one interface element or it cannot participate in any
topology.
    Thus, for each SAAM node, there should be one or more Interface elements.
4· Each Interface element should have one Address element and one MaskBits element.
5· Each node can have zero or more Agent elements.

These rules are represented by the following DTD syntax

Operator    Description
No Operator      Must appear exactly one time
?           Must appear once or not at all
+           Must appear at least once ( 1 ... N times)
*           May appear any number of times, or not at all ( 0 ... N times)
-->
<!-- PCDATA stands for Parsed Character Data and it means that the element contains text data "
-->
<!ELEMENT Address (#PCDATA)>
<!ELEMENT AgentName (#PCDATA)>
<!ELEMENT Param (#PCDATA)>
<!ELEMENT Agent (AgentName, Param*)>
<!ELEMENT IPv4 (#PCDATA)>
<!ELEMENT Interface (Address, MaskBits, Bandwidth)>
```

```
<!ELEMENT InterfaceAddressType (#PCDATA)>
<!ELEMENT MaskBits (#PCDATA)>
<!ELEMENT Bandwidth (#PCDATA)>
<!ELEMENT Node (NodeType, NodeName, IPv4, ServerFlowID?, TimeScale, SC_MetricType?,
SC_CycleTime?, SC_GlobalWaitTime?, SC_LocalWaitTime, InterfaceAddressType, Interface+,
Agent*)>
<!ELEMENT NodeName (#PCDATA)>
<!ELEMENT NodeType (#PCDATA)>
<!ELEMENT SAAM_Net (Node+)>
<!ELEMENT SC_CycleTime (#PCDATA)>
<!ELEMENT SC_GlobalWaitTime (#PCDATA)>
<!ELEMENT SC_LocalWaitTime (#PCDATA)>
<!ELEMENT SC_MetricType (#PCDATA)>
<!ELEMENT ServerFlowID (#PCDATA)>
<!ELEMENT TimeScale (#PCDATA)>
```

APPENDIX B. SAAM CONFIGURATION XML FILE

```xml
<?xml version="1.0" encoding="UTF-8"?>
<! -- Edited with XML Spy v3.0.7 NT (http://www.xmlspy.com) -->
<!DOCTYPE SAAM_Net SYSTEM "./SAAM_2.dtd">
<! -- The XML file contains the configuration information of the desired network topology. It makes
sure that the structure is correct by validating these contents with the DTD file, which is defined
above.
The SAAM configuration consists from elements, which can contain other elements. The main
element is the node, which represents the servers and routers. Each node has a set of elements and
values. -->

<SAAM_Net>
   <Node>
      <NodeType>PrimaryServer</NodeType>
      <NodeName>Server A</NodeName>
      <IPv4>131.120.8.155</IPv4>
      <ServerFlowID>1</ServerFlowID>
      <TimeScale>350</TimeScale>
      <!--SC = Self Configuration-->
      <SC_MetricType>0</SC_MetricType>
      <SC_CycleTime>200</SC_CycleTime>
      <SC_GlobalWaitTime>250</SC_GlobalWaitTime>
      <SC_LocalWaitTime>0</SC_LocalWaitTime>
      <InterfaceAddressType>IPv6</InterfaceAddressType>
      <Interface>
         <Address>99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.1</Address>
         <MaskBits>40</MaskBits>
         <Bandwidth>155000000</Bandwidth>
      </Interface>
   </Node>
   <Node>
      <NodeType>Router</NodeType>
      <NodeName>Router A</NodeName>
      <IPv4>131.120.8.155</IPv4>
      <TimeScale>350</TimeScale>
      <SC_LocalWaitTime>0</SC_LocalWaitTime>
      <InterfaceAddressType>IPv6</InterfaceAddressType>
      <Interface>
         <Address>99.99.99.99.1.0.0.0.0.0.0.0.0.0.0.2</Address>
         <MaskBits>40</MaskBits>
         <Bandwidth>155000000</Bandwidth>
```

```xml
        </Interface>
        <Interface>
            <Address>99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.1</Address>
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Interface>
            <Address>99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.1</Address>
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Interface>
            <Address>99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1</Address>
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Agent>
            <AgentName>FlowGenerator</AgentName>
            <Param>99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.1</Param>
            <!-- Destination Ipv6 address-->
            <Param>6000</Param>
            <!-- Destination Port number-->
            <Param>IntServ</Param>
            <!-- Type of Service-->
            <Param>200</Param>
            <!-- initial wait -->
            <Param>60</Param>
            <!-- Test Duration-->
            <Param>CBR</Param>
            <!-- Distribution-->
            <Param>500</Param>
            <!-- Packet rate per second-->
            <Param>1410</Param>
            <!-- Payload Size-->
        </Agent>
        <Agent>
            <AgentName>FlowSink</AgentName>
            <Param>6000</Param>
            <!-- receiver port number-->
        </Agent>
    </Node>
    <Node>
        <NodeType>Router</NodeType>
        <NodeName>Router B</NodeName>
```

64

```xml
<IPv4>131.120.8.155</IPv4>
<TimeScale>350</TimeScale>
<SC_LocalWaitTime>0</SC_LocalWaitTime>
<InterfaceAddressType>IPv6</InterfaceAddressType>
<Interface>
    <Address>99.99.99.99.2.0.0.0.0.0.0.0.0.0.0.2</Address>
    <MaskBits>40</MaskBits>
    <Bandwidth>622000000</Bandwidth>
</Interface>
<Interface>
    <Address>99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1</Address>
    <MaskBits>40</MaskBits>
    <Bandwidth>622000000</Bandwidth>
</Interface>
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2</Param>
    <!-- Destination Ipv6 address-->
    <Param>6000</Param>
    <!-- Destination Port number-->
    <Param>IntServ</Param>
    <!-- Type of Service-->
    <Param>200</Param>
    <!-- initial wait -->
    <Param>70</Param>
    <!-- Test Duration-->
    <Param>Packet-Train</Param>
    <!-- Distribution-->
    <Param>40</Param>
    <!-- Packet rate per second-->
    <Param>1420</Param>
    <!-- Payload Size-->
    <Param>5</Param>
    <!-- Average Train Size-->
</Agent>
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1</Param>
    <!-- Destination Ipv6 address-->
    <Param>6000</Param>
    <!-- Destination Port number-->
    <Param>IntServ</Param>
    <!-- Type of Service-->
    <Param>200</Param>
```

```xml
            <!-- initial wait -->
            <Param>60</Param>
            <!-- Test Duration-->
            <Param>CBR</Param>
            <!-- Distribution-->
            <Param>500</Param>
            <!-- Packet rate per second-->
            <Param>1410</Param>
            <!-- Payload Size-->
        </Agent>
        <Agent>
            <AgentName>FlowSink</AgentName>
            <Param>6000</Param>
            <!-- receiver port number-->
        </Agent>
    </Node>
    <Node>
        <NodeType>Router</NodeType>
        <NodeName>Router C</NodeName>
        <IPv4>131.120.8.152</IPv4>
        <TimeScale>350</TimeScale>
        <SC_LocalWaitTime>0</SC_LocalWaitTime>
        <InterfaceAddressType>IPv6</InterfaceAddressType>
        <Interface>
            <Address>99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.2</Address>
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Interface>
            <Address>99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.1</Address>
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Agent>
            <AgentName>FlowSink</AgentName>
            <Param>6000</Param>
            <!-- receiver port number-->
        </Agent>
    </Node>
    <Node>
        <NodeType>Router</NodeType>
        <NodeName>Router D</NodeName>
        <IPv4>131.120.8.152</IPv4>
        <TimeScale>350</TimeScale>
```

66

```xml
        <SC_LocalWaitTime>0</SC_LocalWaitTime>
        <InterfaceAddressType>IPv6</InterfaceAddressType>
        <Interface>
            <Address>99.99.99.99.6.0.0.0.0.0.0.0.0.0.0.2</Address>
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Interface>
            <Address>99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1</Address>
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Agent>
            <AgentName>FlowGenerator</AgentName>
            <Param>99.99.99.99.5.0.0.0.0.0.0.0.0.0.0.1</Param>
            <!-- Destination Ipv6 address-->
            <Param>7000</Param>
            <!-- Destination Port number-->
            <Param>IntServ</Param>
            <!-- Type of Service-->
            <Param>200</Param>
            <!-- initial wait -->
            <Param>70</Param>
            <!-- Test Duration-->
            <Param>Packet-Train</Param>
            <!-- Distribution-->
            <Param>40</Param>
            <!-- Packet rate per second-->
            <Param>1420</Param>
            <!-- Payload Size-->
            <Param>5</Param>
            <!-- Average Train Size-->
        </Agent>
        <Agent>
            <AgentName>FlowSink</AgentName>
            <Param>7000</Param>
            <!-- receiver port number-->
        </Agent>
        <Agent>
            <AgentName>FlowSink</AgentName>
            <Param>6000</Param>
            <!-- receiver port number-->
        </Agent>
    </Node>
```

```xml
<Node>
    <NodeType>Router</NodeType>
    <NodeName>Router E</NodeName>
    <IPv4>131.120.8.152</IPv4>
    <TimeScale>350</TimeScale>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
    <InterfaceAddressType>IPv6</InterfaceAddressType>
    <Interface>
        <Address>99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.2</Address>
        <MaskBits>40</MaskBits>
        <Bandwidth>622000000</Bandwidth>
    </Interface>
    <Interface>
        <Address>99.99.99.99.8.0.0.0.0.0.0.0.0.0.0.1</Address>
        <MaskBits>40</MaskBits>
        <Bandwidth>622000000</Bandwidth>
    </Interface>
    <Agent>
        <AgentName>FlowGenerator</AgentName>
        <Param>99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.1</Param>
        <!-- Destination Ipv6 address-->
        <Param>6000</Param>
        <!-- Destination Port number-->
        <Param>IntServ</Param>
        <!-- Type of Service-->
        <Param>200</Param>
        <!-- initial wait -->
        <Param>60</Param>
        <!-- Test Duration-->
        <Param>CBR</Param>
        <!-- Distribution-->
        <Param>500</Param>
        <!-- Packet rate per second-->
        <Param>1410</Param>
        <!-- Payload Size-->
    </Agent>
    <Agent>
        <AgentName>FlowSink</AgentName>
        <Param>6000</Param>
        <!-- receiver port number-->
    </Agent>
</Node>
<Node>
    <NodeType>Router</NodeType>
```

```xml
<NodeName>Router F</NodeName>
<IPv4>131.120.8.155</IPv4>
<TimeScale>350</TimeScale>
<SC_LocalWaitTime>0</SC_LocalWaitTime>
<InterfaceAddressType>IPv6</InterfaceAddressType>
<Interface>
    <Address>99.99.99.99.8.0.0.0.0.0.0.0.0.0.0.2</Address>
    <MaskBits>40</MaskBits>
    <Bandwidth>622000000</Bandwidth>
</Interface>
<Interface>
    <Address>99.99.99.99.4.0.0.0.0.0.0.0.0.0.0.2</Address>
    <MaskBits>40</MaskBits>
    <Bandwidth>622000000</Bandwidth>
</Interface>
<Agent>
    <AgentName>FlowGenerator</AgentName>
    <Param>99.99.99.99.7.0.0.0.0.0.0.0.0.0.0.1</Param>
    <!-- Destination Ipv6 address-->
    <Param>7000</Param>
    <!-- Destination Port number-->
    <Param>IntServ</Param>
    <!-- Type of Service-->
    <Param>200</Param>
    <!-- initial wait -->
    <Param>70</Param>
    <!-- Test Duration-->
    <Param>Poisson</Param>
    <!-- Distribution-->
    <Param>300</Param>
    <!-- Packet rate per second-->
    <Param>1430</Param>
    <!-- Payload Size-->
</Agent>
</Node>
<Node>
    <NodeType>Router</NodeType>
    <NodeName>Router G</NodeName>
    <IPv4>131.120.8.152</IPv4>
    <TimeScale>350</TimeScale>
    <SC_LocalWaitTime>0</SC_LocalWaitTime>
    <InterfaceAddressType>IPv6</InterfaceAddressType>
    <Interface>
        <Address>99.99.99.99.3.0.0.0.0.0.0.0.0.0.0.2</Address>
```

```xml
            <MaskBits>40</MaskBits>
            <Bandwidth>622000000</Bandwidth>
        </Interface>
        <Agent>
            <AgentName>FlowGenerator</AgentName>
            <Param>99.99.99.99.8.0.0.0.0.0.0.0.0.0.0.1</Param>
            <!-- Destination Ipv6 address-->
            <Param>6000</Param>
            <!-- Destination Port number-->
            <Param>IntServ</Param>
            <!-- Type of Service-->
            <Param>200</Param>
            <!-- initial wait -->
            <Param>60</Param>
            <!-- Test Duration-->
            <Param>CBR</Param>
            <!-- Distribution-->
            <Param>500</Param>
            <!-- Packet rate per second-->
            <Param>1410</Param>
            <!-- Payload Size-->
        </Agent>
        <Agent>
            <AgentName>FlowSink</AgentName>
            <Param>6000</Param>
            <!-- receiver port number-->
        </Agent>
    </Node>
</SAAM_Net>
```

APPENDIX C. SAXPARSERDEMO CLASS SOURCE CODE

```
// Dec2000 [Fatih] added sent agent parameters
// Nov2000.[Fatih] chanced to run on multiple Instances per host
// 10 August 00 [Mohammad Ababneh] created

package saam.demo;

import saam.*;
import saam.control.*;
import saam.message.*;
import saam.agent.*;
import saam.router.*;
import saam.net.*;
import saam.util.*;
import saam.demo.*;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;

// Apache Xerces XML-SAX parser imports
import java.io.IOException;

import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.Locator;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.XMLReaderFactory;


/**
 * Class   : SAXParserDemo * Purpose: will take an XML file and parse it
 * using SAX. the XML file
 * contains the configuration information of the SAAM servers and
 * routers.
 *
 * Program : SAXParserDemo.java
 * Updated : Sep 6, 2000
 * Author  : Mohammad Ababneh
 * Purpose : This Demo Station Configuration program will setup the
 *      topology for the SAAM network. It is reading the configuration
 *      information from an XML file that can be selected by the user.
 *      The XML file contains a special configuration meta-data
 *      developed for this purpose.
 *      There is no need to hardcode the configuration information inside
```

71

```java
 *       the demo station programs.
 */
public class SAXParserDemo
{
  /**
    * Method      : performDemo
    * Purpose     : This method parses the file, using registered SAX
    *               handlers, and output
    *               the events in the parsing process cycle.
    * Parameters : uri  ( String uri ) URI of the file to parse.
    */

  public void performDemo(String uri, String answer)
  {
    System.out.println("Parsing XML File: " + uri + "\n\n");
    // Get instances of the major classes needed
    ContentHandler contentHandler;
    if (answer.equals("1"))
    {
      contentHandler = new MyContentHandler();
    }
    else
    {
      contentHandler = new MyContentHandlerII();
    }
    ErrorHandler    errorHandler   = new MyErrorHandler();
    try
    {
      // Instantiate a SAX parser
      XMLReader parser = XMLReaderFactory.createXMLReader(
                    "org.apache.xerces.parsers.SAXParser");
      // Register the content handler
      parser.setContentHandler(contentHandler);
      // Register the error handler
      parser.setErrorHandler(errorHandler);
      // Parse the document in the location uri
      // it can be local or in a URL
      parser.parse(uri);
    }
    catch (IOException e)
    {
      System.out.println("Error reading URI: " + e.getMessage());
    }
    catch (SAXException e)
    {
      System.out.println("Error in parsing: " + e.getMessage());
    }
  } // end performDemo
```

```java
/**
 * Method      : main
 * Purpose     : The main method in the SAXParserDemo class.
 * Parameters : String[] args
 */
public static void main(String[] args)
{
  DemoGui guiContainer = new DemoGui("SAAM XML SAX Parser DemoStation
     ver 1.0    August 2000 ");
  StringBuffer uri = new StringBuffer("file:///");

  guiContainer.sendText(" (1) NEW NETWORK CONFIGURATION");
  guiContainer.sendText(" (2) JUST SEND AGENT(S) TO ROUTERS ");
  String answer =
    JOptionPane.showInputDialog( "Choose one of them? (1) or (2) " );

  if (answer != null)
  {
    //display the File Chooser here
    // open the file through a file chooser object and read data

    File currentFile =
       new File(FileIO.getWorkingDir() + "\\demo");
       //starting directory -crc
    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);

//      Vector /*passableFileSuffixes*/ pfs = new Vector(); //Reject all
//      file types except these
//      pfs.add(".xml"); pfs.add(".txt"); pfs.add(".dtd");
//      fileChooser.setFileFilter(
//      new ConfigFileFilter(pfs, // suffix pass list
//
//    "SAAM_2", //further qualify selection on this filename preamble
//    "SAAM Config Files"  // display this on chooser window
//                                  ) ); // install filename filter -crc


      if (answer.equals("1"))  //new network configuration (with
                               //agents)
      {
        fileChooser.setFileFilter(
           new ConfigFileFilter(".xml", // Single suffix passed
           "xml", //further qualify selection on this filename preamble
           "SAAM Config XML Files"  // display this on chooser window
                             ) ); // install filename filter -crc
      }
      else if (answer.equals("2")) //send just agent(s)
      {
        fileChooser.setFileFilter(
           new ConfigFileFilter(".xml", // Single suffix passed
           "sendAgent", //further qualify selection on this filename
                          preamble
```

73

```
                      "SAAM Config XML Files"  // display this on chooser window
                                  ) ); // install filename filter -crc
        }

        fileChooser.setCurrentDirectory(currentFile);
        int result = fileChooser.showOpenDialog(guiContainer);
        // display the dialog box in the screen center
        // Associate Dialog object with the gui panel fixes panel hiding
        // errors -crc

        if (result != JFileChooser.APPROVE_OPTION)
        //fix cancel button action and close frame action -crc
          System.exit(0);
        File inFile = fileChooser.getSelectedFile();
        // format the location of the XML file as an URI by adding the
         path to "file:///"
        uri.append(inFile.getPath());
        System.out.println(" URI of the XML file is : " + uri);
        // instantiate a new SAXParserDemo object
        SAXParserDemo parserDemo = new SAXParserDemo();
        parserDemo.performDemo(uri.toString(), answer);
        //call the performDemo method that will call the XML content
         handler
        //parserDemo.performDemo(uri.toString());
        System.out.println("\nNetwork setup is completed; This GUI will
                            disappear in 30 minutes.");

        try
        {
          Thread.sleep(30000); //sleep 30 seconds
        }
        catch(InterruptedException ie)
        {
          //gui.sendText("problem after initrouter thread sleep");
          System.out.println("problem after initrouter thread sleep");
        }
      }
      System.exit(0);
   }   // end main method
} // end SAXParserDemo


/**
 * Class   : MyContentHandler
 * Purpose: implements the SAX interface and parses the document
 *          based on the events that happen inside the document
 *          it establishes the basic data structures that gather the
 *          configuration information from the XML file.
 */
class MyContentHandler implements ContentHandler
{
  private ObjectOutputStream output ;
  RouterRecord record;
```

74

```java
public DemoGui gui = new DemoGui("SAAM XML SAX Parser DemoStation  ver
                                 1.0    August 2000 ");

// Hold onto the locator for location information
private Locator locator;
// all global variables
// temp variables to store values read from the XML file
public String element = "" , value ="";
// element & value
String tIPv4 ="" , tSourceInterfaceAddress ="",
                   tInterfaceAddressType="";
String tMaskBits = "", tBandwidth="";
String tAgentName ="" , tParam ="" ;
String tNodeName ="" , tNodeType ="";

int tServerType = 5, tServerFlowID = 5,tTimeScale = 5;
int tSC_MetricType = 5, tSC_CycleTime = 5 , tSC_GlobalWaitTime = 5,
    tSC_LocalWaitTime = 5;

String [] []  interfaceArray = new String [100][7];
// array of interface objects:
String [] []  agentArray = new String [100][20];
// array of agent objects
int interfaceCount   = 0;
// the total number of interfaces in the interface array
int interfaceIndex   = 0;
// interfaceArray index
int agentArrayLength = 0;
// index of agents in the agent array
int paramCounter     = 4;
// index of number of parametersper   agent
DemoInitInfo[] router  = new DemoInitInfo[100];
// array of router objects
DemoInitInfo[] bServer = new DemoInitInfo[10];
// array of backup objects
DemoInitInfo[] mServer = new DemoInitInfo[1];
// array of server objects
InterfaceID [] interfaceIP6  = new InterfaceID[100];
// array of InterfaceIP6 objects
int routerCount = 0; // router sequential counter
int backupCount = 0; // backup sequential server counter
boolean backupExist = false;
// variable to decide if a backup server exists
boolean newNode = false;  // variable to decide if this is a new node
EmulationPort v4Port = new EmulationPort(9002); //   fatih
String whoAmI = "uninitialized";
// Player's name string built from config file parsing -crc


/**
* Method     : setDocumentLocator
* Purpose    : This method gives the capability to define the exact
               location while
```

75

```java
*              parsing the XML file
* Parameters : Locator locator
*/
public void setDocumentLocator(Locator locator)
{
  System.out.println("    * setDocumentLocator() called");
  // We save this for later use if desired.
  this.locator = locator;
}


/**
 * Method      : startDocumnet
 * Purpose     : In this method there can be any kind of statements
 * that we would like to occur when we first open an XML document
 * Parameters :
 * throws      : SAXException when things go wrong
*/
public void startDocument() throws SAXException
{
  System.out.println("Parsing begins...");
  try
  {
    output = new ObjectOutputStream(
             new FileOutputStream( "RouterInfo.txt" ) );
  }
  catch(IOException ioe)
  {
    gui.sendText(ioe.toString());
  }
}


/**
 * Method      : startElement
 * Purpose     : This reports the occurrence of an actual element.
                 Itwill include the element's attributes, with the
 * exception ofXML vocabulary
 * specific attributes like "DTD",....
 * Parameters : String namespaceURI, String localName,
 *              String rawName, Attributes atts)
 * throws      : SAXException when things go wrong
 */
public void startElement(String namespaceURI, String localName,String
                         rawName, Attributes atts)
  throws SAXException
{
  System.out.print(" \n startElement: " + localName);
  element = localName;
}
```

```java
/**
 * Method      : characters
 * Purpose     : This method returns the real value stored in the
 *               XMLelement
 *               It is then converted to a string " array of characters
 * Parameters  : char[] ch, int start, int end
 * throws      : SAXException when things go wrong
 */
public void characters(char[] ch, int start, int end)throws
    SAXException
{
  String s = new String(ch, start, end);
  value = s;
}


/**
 * Method      : endElement
 * Purpose     : Indicates the end of an element is reached.  Note that
 *   the parser does not distinguish between empty elements and
 *   non-empty elements, so this will occur uniformly.
 *   We gather the value of the element when we reach the end
 *   of the element
 * Parameters  : String namespaceURI, String localName, StringrawName
 * throws      : SAXException when things go wrong
 */
public void endElement(String namespaceURI, String localName,String
                       rawName)
  throws SAXException
{
  System.out.println("\n element = " + element + "   value =  " +
                     value);
  if (element.equals("NODE"))
  {
    newNode  = true;    // this is a new node
    gui.sendText("== The beginning of a new node  =============");
  }
  else if  (element.equals("NodeType"))
  {
    tNodeType = value;
    if (tNodeType.startsWith("P"))
    {
      tServerType=0;
    }
    else  if(tNodeType.startsWith("B"))
    {
      tServerType=1;
    }
    gui.sendText("========= The beginning of a new node ===========");
    gui.sendText(" Node Type : " + tNodeType);
  }
  else if  (element.equals("NodeName"))
  {
```

```java
    tNodeName = value;
    gui.sendText(" Node Name : " + tNodeName);
}
else if    (element.equals("IPv4"))
{
    tIPv4 = value;
    gui.sendText(" IPv4 : " + tIPv4);
    v4Port = newEmulationPort(tIPv4, router, bServer, mServer);
}
else if (element.equals("ServerFlowID"))
{
    tServerFlowID = Integer.parseInt(value);
    gui.sendText(" ServerFlowID : " + tServerFlowID);
}
else if (element.equals("TimeScale"))
{
    tTimeScale = Integer.parseInt(value);
    gui.sendText(" TimeScale : " + tTimeScale);
}
else if (element.equals("SC_MetricType"))
{
    tSC_MetricType = Integer.parseInt(value);
    gui.sendText(" SC_MetricType : " + tSC_MetricType);
}
else if (element.equals("SC_CycleTime"))
{
    tSC_CycleTime = Integer.parseInt(value);
    gui.sendText(" SC_CycleTime : " + tSC_CycleTime);
}
else if (element.equals("SC_GlobalWaitTime"))
{
    tSC_GlobalWaitTime = Integer.parseInt(value);
    gui.sendText(" SC_GlobalWaitTime : " + tSC_GlobalWaitTime);
}
else if (element.equals("SC_LocalWaitTime"))
{
    tSC_LocalWaitTime = Integer.parseInt(value);
    gui.sendText(" SC_LocalWaitTime : " + tSC_LocalWaitTime);
}
else if (element.equals("InterfaceAddressType"))
{
    tInterfaceAddressType = value;
    gui.sendText(" InterfaceAddressType : " + tInterfaceAddressType);
}
else if (element.equals("Interface"))
{
    boolean newInterface = true;
    gui.sendText(" A new interface was discovered");
}
else if (element.equals("Address"))
{
    tSourceInterfaceAddress = value.toString();
    gui.sendText("sourceInterfaceAddress:" + tSourceInterfaceAddress);
```

78

```java
    }
    else if (element.equals("MaskBits"))
    {
      tMaskBits = value;
      gui.sendText(" MaskBits : " + tMaskBits);
    }
    else if (element.equals("Bandwidth"))
    {
      tBandwidth = value;
      gui.sendText(" MaskBits : " + tMaskBits);
    }

    // at the end of Interface element, add an entry to the
       Interfacearray
    if (localName.equals("Bandwidth"))
    {
      gui.sendText(" Add a row to the interface matrix");
      // fill the interface address array
      interfaceArray[interfaceIndex][0] = tNodeType;
      interfaceArray[interfaceIndex][4] = tNodeName;
      interfaceArray[interfaceIndex][2] = tSourceInterfaceAddress;
      interfaceArray[interfaceIndex][5] = tMaskBits;
      interfaceArray[interfaceIndex][6] = tBandwidth;

      if(interfaceArray[interfaceIndex][0].startsWith("R"))
      // routerinterface
      {
        String str = Integer.toString(routerCount);
        interfaceArray[interfaceIndex][1] = str;
      }
      else if(interfaceArray[interfaceIndex][0].startsWith("B"))
      //backup server interface
      {
        String str = Integer.toString(backupCount);
        interfaceArray[interfaceIndex][1] = str;
      }
      interfaceIndex++;      // increase number of interfaces
    } //end if (localName.equals("MaskBits"))

    if (element.equals("Agent"))
    {
      gui.sendText("========= The beginning of an Agent  ============");
    }
    else if (element.equals("AgentName"))
    {
      tAgentName = value;
      gui.sendText(" AgentName : " + tAgentName);
    }
    else if (element.equals("Param"))
    {
      tParam = value;
      gui.sendText(" Param " + paramCounter +" : " + tParam);
    }
```

79

```java
if (localName.equals("AgentName"))
{
  agentArray[agentArrayLength][3] = tAgentName; // agent name
}
if (localName.equals("Param"))
{
  agentArray[agentArrayLength][paramCounter] = tParam;
  paramCounter++;
}
else if (localName.equals("Agent"))
{
  // at the end of Agent element, add an entry to the Agent array
  gui.sendText(" Add the agent to the agent array ... ");
  agentArray[agentArrayLength][1] = tIPv4;  //node IP Address;
  agentArray[agentArrayLength][2] = v4Port.toString();
  // Emulationport -- [fatih]
  agentArray[agentArrayLength][0] = tNodeName; // router name
  paramCounter = 4;
  agentArrayLength++;    // increase number of agents
}

//   Start creating the node once all data is collected
//   When finding the end of the node </node>
try
{
  gui.setTextField("MyIP:
        "+InetAddress.getLocalHost().getHostAddress());
  if (localName.equals("Node"))
  {
    gui.sendText("======== The end of the node ==================");
    whoAmI = new String(tNodeName);   // fatih
    if (tNodeType.startsWith("P"))
    {
      // creating a primary server node
      Configuration cfMainServer = new Configuration(
                          (byte)tServerType,
                          tServerFlowID,
                          (byte)tSC_MetricType,
                          tSC_CycleTime,
                          tSC_GlobalWaitTime);
  mServer[0]=  new DemoInitInfo (gui, InetAddress. GetByName (tIPv4)
      , v4Port,tTimeScale,whoAmI,cfMainServer);   // fatih
      gui.sendText(" the server is created , and its IP is :   " +
            InetAddress.getByName(tIPv4));
    }
    else if (tNodeType.startsWith("B"))
    {
    // creating a backup server node for as many backup server as
       there are
       backupExist = true;
       Configuration cfBackupServer = new Configuration(
                          (byte)tServerType,
```

80

```
                              tServerFlowID,
                              (byte)tSC_MetricType,
                              tSC_CycleTime,
                              tSC_GlobalWaitTime);
        bServer[backupCount]=new DemoInitInfo(gui,InetAddress.getByName
              (tIPv4),v4Port,tTimeScale,whoAmI,cfBackupServer);
       gui.sendText(" the backup server is created , and its IP is :   "
              + InetAddress.getByName(tIPv4));
        backupCount++;
      } // end backup server
      else
      {
         // creating a router node
         router[routerCount]=newDemoInitInfo gui,InetAddress.getByName
           (tIPv4),v4Port,tTimeScale,whoAmI);
         record = new RouterRecord(whoAmI, tIPv4 , v4Port.toString());
      //    gui.sendText(" record routerName: " +
                 record.nodeName);
      // gui.sendText(" record ipv4        : " + record.ipv4Address);
      // gui.sendText(" record v4port     : " + record.v4Port);
      // writes parameters to the file those are neded if we want to
         sent
      // agents to the nodes on the fly.
      try
      {
         output.writeObject( record );
      }
      catch ( IOException io )
      {
        gui.sendText(io.toString());
      }
      gui.sendText(" a new router is created , and its IP is :   " +
            InetAddress.getByName(tIPv4));
      routerCount++
     }   // end router
   } // end if
  }//try block

  catch(UnknownHostException uhe)
  {
    gui.sendText(uhe.toString());
  }
}   // end endElement method


/**
 * Method      : endDocument
 * Purpose     : This method indicates the end of the XML document.
 *   it is reached when it finishesall the parsing events inside that
 *   we would like to occur when we first open an XML document
 * Parameters : None
 * throws      : SAXException when things go wrong
 */
```

81

```java
public void endDocument() throws SAXException
{
  // close file before the program ends
  try
  {
    output.close();
  }
  catch ( IOException e )
  {
    gui.sendText(e.toString());
    gui.sendText("Error closing File");
  }
  System.out.println("...Parsing ends.");
  gui.sendText("================================================");
  gui.sendText("Start creating interfaces when reaching the end of the
    document");
  gui.sendText("================================================");
  int routerCount = 0; // counter for router in interface array
  int backupCount = 0; // counter for backup servers in interface
                          array
  // start creating interfaces on the nodes
  for (int idx = 0 ; idx < interfaceIndex ; idx++)
  {
    if (interfaceArray[idx][0].startsWith("P"))
    {
      interfaceIP6[idx] = mServer[0].getNewInterface
      (interfaceArray[idx][2],
      Integer.parseInt(interfaceArray[idx][6]),
      Integer.parseInt(interfaceArray[idx][5]));
      gui.sendText("creating primary server interface number :" +
      idx);
      System.out.println("creating server interface number :" + idx);
     }
    else if (interfaceArray[idx][0].startsWith("B"))
    {
      backupCount = Integer.parseInt(interfaceArray[idx][1]);
      interfaceIP6[idx]=bServer[backupCount] .getNewInterface
    (interfaceArray[idx][2], Integer.parseInt(interfaceArray[idx][6]),
     Integer.parseInt(interfaceArray[idx][5]));
     gui.sendText("creating backup server interface number : " + idx);
     System.out.println("creating backup server interface number :" +
     idx);
    }
    else
    {
      routerCount = Integer.parseInt(interfaceArray[idx][1]);
      interfaceIP6[idx]
          router[routerCount].getNewInterface(interfaceArray[idx][2],
          Integer.parseInt(interfaceArray[idx][6])
          Integer.parseInt(interfaceArray[idx][5]));
      gui.sendText("creating router interface number :   " + idx);
    }// end else
    interfaceCount = idx;    // number of interfaces in the array
```

82

```
      } // end for
      // find which interfaces are on the same network and sharing the
        same link
      // by finding the network address and comparing the two interface
addresses
      for (int i = 0; i <= interfaceCount; i++)
      {
        try
        {
          byte tByteI = Byte.parseByte (interfaceArray[i][5]);
          for ( int j = i + 1; j<=interfaceCount; j++)
          {
            byte tByteJ = Byte.parseByte (interfaceArray[j][5]);
            if ((interfaceIP6[i].getIPv6().getNetworkAddress(tByteI)).
            equals(interfaceIP6[j].getIPv6().getNetworkAddress(tByteI)))
                {
                  gui.sendText("--- equal interface  "+
                   interfaceIP6[i].getIPv6() +" with interface " +
                  interfaceIP6[j].getIPv6());
                  // find the destination IPv6 address for the source
                   IPv6  address
                  interfaceArray[i][3]=interfaceIP6[j].getIPv6().
                   toString();
                  interfaceArray[j][3]=interfaceIP6[i].getIPv6().
                   toString();
                }
            }
        }
        catch(NumberFormatException ce)
        {
          gui.sendText(ce.toString());
        }
      }
      // display the interface and agent arrays
      gui.sendText(" /////////////////////////////////////////////");
      gui.sendText(" // display the interface array.... which has  " +
       interfaceIndex + "  interfaces");
      for (int idx = 0; idx < interfaceIndex; idx++)
      {
       gui.sendText(interfaceArray[idx][0]+ "/" + interfaceArray[idx][1]
+ "/" +interfaceArray[idx][2] + "/" +interfaceArray[idx][3] + "/"+
       interfaceArray[idx][4]+ "/" + interfaceArray[idx][5] );
       }
      gui.sendText("// display the agent array.... which has  " +
      agentArrayLength + "  agents");
      for (int idx = 0; idx < agentArrayLength; idx++)
      {
        gui.sendText(agentArray[idx][0]+ "/" + agentArray[idx][1] + "/"
        + agentArray[idx][2]+ "/" +
        agentArray[idx][3]+ "/" + agentArray[idx][4] );
      }
      // connecting the Interfaces
      String sourceIP, destIP = "";   // source IPv6 and destination IPv6
```

```java
for (int sourceCount=0; sourceCount <= interfaceCount;
     sourceCount++)
{
    sourceIP = interfaceArray[sourceCount][2];
    // look for the corresponding interface to link with
    for (int destCount=0; destCount <= interfaceCount; destCount++)
    {
      destIP = interfaceArray[destCount][3]; // was [2] - crc
      if (sourceIP.equals(destIP))
      {
        if (interfaceArray[sourceCount][0].startsWith("P")) //
            check the node type
        {
          // call the connect method
          mServer[0].isConnectedTo((interfaceIP6[destCount]));
        }
        else if (interfaceArray[sourceCount][0].startsWith("B")) //
                  check the node type
        {
          int backupIndex =
          Integer.parseInt(interfaceArray[sourceCount][1]);
          // call the connect method
          bServer[backupIndex].isConnectedTo
          ((interfaceIP6[destCount]));
        }
        else if (interfaceArray[sourceCount][0].startsWith("R"))
        // check the node type
        {
          int routerIndex =
           Integer.parseInt(interfaceArray[sourceCount][1]);
          //call the connect method
          router[routerIndex].isConnectedTo
          ((interfaceIP6[destCount]));
        }
        gui.sendText("connecting node :   " +
          interfaceArray[sourceCount][4] +" with " +
          interfaceArray[destCount][4]);
      } // end if
    }  // end for
  }  // end for

  // Activating the nodes
  // use a loop to activate server, backup servers and routers
  // activate the server
  mServer[0].activate(gui);
  gui.sendText("Activating the Server" );
  // activate the routers
  for (int idx = 0; idx <= routerCount; idx++)
  {
    router[idx].activate(gui);
    gui.sendText("Activating the router : " + router[idx] );
  }
```

84

```java
      // activate the backup servers
      if (backupExist)
      {
        for (int idx = 0; idx <= backupCount; idx++)
        {
          bServer[idx].activate(gui);
        }
        backupExist = false;   // there is no backup servers
      }
      // sending the agents to the nodes
      StringBuffer agentStringBuffer = new StringBuffer("");
      for (int agentCounter = 0; agentCounter < agentArrayLength;
            agentCounter++)
      {
        agentStringBuffer.append("saam.agent.applications.");
        // the place where application agents are
        agentStringBuffer.append(agentArray[agentCounter][3]);
        // agent name
        DemoInitInfo.sendAgent(gui,agentArray[agentCounter]
          ,agentStringBuffer.toString());
        agentStringBuffer.delete(0,agentStringBuffer.length());
        // clear the string buffer to hold a new agent
      } // end for
} // end of endDocument Method


/**
 * Method      : processingInstruction
 * Purpose     : This method is used when processing instructions
 * are found
 * in the XML file. The current XML file doesn't support PI
 * Parameters : String target, String data
 * throws      : SAXException when things go wrong
 */
public void processingInstruction(String target, String
    data)throws SAXException
{
  System.out.println("PI: Target:" + target + " and Data:" +
    data);
}


/**
 * Method      : startPrefixMapping
 * Purpose     : This method is used in case XML name spaces are
 *   used.
 * In the current use of XML. name spaces are not supported.
 * Parameters : String prefix, String uri
 * throws      : SAXException when things go wrong
 */
public void startPrefixMapping(String prefix, String uri)
{
  System.out.println("Mapping starts for prefix " + prefix +
                      " mapped to URI " + uri);
```

```java
}

/**
 * Method      : endPrefixMapping
 * Purpose     : This method is used in case XML name spaces are
 *   used.
 * In the current use of XML. name spaces are not supported.
 * Parameters : String prefix, String uri
 * throws      : SAXException when things go wrong
 */
public void endPrefixMapping(String prefix)
{
  System.out.println("Mapping ends for prefix " + prefix);
}


/**
 * Method      : characters
 * Purpose     : This method provides the ability to report white
 * spaces
 * Parameters : char[] ch, int start, int end
 * throws      : SAXException when things go wrong
 */
public void ignorableWhitespace(char[] ch, int start, int
    end)throws SAXException
{
  String s = new String(ch, start, end);
  System.out.println("ignorableWhitespace: [" + s + "]");
}

/**
 * Method      : characters
 * Purpose     : This method will report an entity that is
 *   skipped by the parser.
 * This should only occur for non-validating parsers, and then
 * is still implementation-dependent behavior.
 * Parameters : String name
 * throws      : SAXException when things go wrong
 */
public void skippedEntity(String name) throws SAXException
{
  System.out.println("Skipping entity " + name);
}


// Added to be able to run multiple instances on the same
   machine // crc's code
/**
 * This private method is used to change the current emulation
 * port in-effect if
 * a SAAM player with the same IPV4 address already exists.
 * This player must be
 * another instance running on the same host, and hence must
```

```
 * use a unique emulation
 * port on that machine.  When called, if other player instances
 *  are known by the demoStation,
 * for a particular host, this instance emulation port member
 * is to be updated with the highest
 * emulation port found for this IPv4 address plus 2.  This
 * assures predictable connection
 * parameters for each player. Method returns the proper port
 * object to be used.
 */
public EmulationPort newEmulationPort(String tIPv4,
        DemoInitInfo[] routerArray
        DemoInitInfo[] bServerArray ,DemoInitInfo[]
         mServerArray) //under development -crc
{
  //requires that searchable lists of IPv4 addresses in-play be
    maintained as the config file
  // is being parsed.
  int emulationPort = 9002; // the default if tIPv4 is unique
      among players thus far.
  int trialPort;
  // first loop through all routers known thus far.
  for( int idx = 0;  idx <= routerArray.length - 1 &&
      routerArray[idx] != null; idx++)
  {
    String IPv4Tag = routerArray[idx].queryIPv4AddrTag();
     //Inspect the DemoInitInfo objects
    if (IPv4Tag.endsWith(tIPv4))  //An instance with this IPv4
          Address was found
    {
      trialPort = routerArray[idx].queryEmulationPort().toInt();
      if(trialPort >= emulationPort)
        emulationPort = trialPort + 2; //potential new port to
        use
    } // end if
  }//end for
  //now loop through backup server(s)
  for( int idx = 0; idx <= bServerArray.length - 1 &&
      bServerArray[idx] != null; idx++)
  {
    String IPv4Tag = bServerArray[idx].queryIPv4AddrTag();
    //Inspect the DemoInitInfo objects
    if (IPv4Tag.endsWith(tIPv4))  //An instance with this IPv4
          Address was found
    {
      trialPort =
         bServerArray[idx].queryEmulationPort().toInt();
      if(trialPort >= emulationPort)
        emulationPort = trialPort + 2;
        //potential new port to use
    } // end if
  } //end for
  //finally, inspect primary server
```

87

```
        for( int idx = 0; idx <= mServerArray.length - 1 &&
             mServerArray[idx] != null; idx++)
    {
      String IPv4Tag = mServerArray[idx].queryIPv4AddrTag();
      //Inspect the DemoInitInfo objects
      if (IPv4Tag.endsWith(tIPv4))
      //An instance with this IPv4 Address was found
      {
        trialPort =
           mServerArray[idx].queryEmulationPort().toInt();
        if(trialPort >= emulationPort)
          emulationPort = trialPort + 2;
        //potential new port to use
      } // end if
    }//end for
    return(new EmulationPort(emulationPort));
  } //end method newEmulationPort
} // end of SAXParserDemo



/**
 * Class   : MyErrorHandler
 * Purpose: implements the SAX ErrorHandler interface.
 *
 */
class MyErrorHandler implements ErrorHandler
{
  /**
   * Method      : warning
   * Purpose     : This method will report a warning that has
   * occurred; this indicates
   * that while no XML rules were "broken", something appears
   * to be incorrect or missing.an entity that is skipped by the
   * parser.
   * Parameters : SAXParseException exception
   * throws      : SAXException when things go wrong
   */
  public void warning(SAXParseException exception)throws
        SAXException
  {
    System.out.println("**Parsing Warning**\n" +
                       "  Line:    " +
                         exception.getLineNumber() + "\n" +
                       "  URI:     " +
                         exception.getSystemId() + "\n" +
                       "  Message: " +
                         exception.getMessage());

    throw new SAXException("Warning encountered");
  }
```

88

```java
/**
 * Method      : error
 * Purpose     : This method will report an error if a non-
 *   critical parsing error
 * has occurred. This error indicates that even if an XML rules
 * was
 * broken, the parsing can  continue.
 * Parameters : SAXParseException exception
 * throws      : SAXException when things go wrong
 */
public void error(SAXParseException exception)throws
        SAXException
{
  System.out.println("**Parsing Error**\n" +
                    "  Line:      " +
                    exception.getLineNumber() + "\n" +
                    "  URI:       " +
                    exception.getSystemId() + "\n" +
                    "  Message: " +
                    exception.getMessage());
  throw new SAXException("Error encountered");
}


/**
 * Method      : fatalError
 * Purpose     : This method will report a fatal error if a
 *    critical parsing
 *   error has occurred. This fatal error indicates that the
 * parsing process can't be continued because of a major
 * violation
 * to XML rules.
 * Parameters : SAXParseException exception
 * throws      : SAXException when things go wrong
 */
public void fatalError(SAXParseException exception)throws
        SAXException
{
  System.out.println("**Parsing Fatal Error**\n" +
                    "  Line:      " +
                    exception.getLineNumber() + "\n" +
                    "  URI:       " +
                    exception.getSystemId() + "\n" +
                    "  Message: " +
                    exception.getMessage());
  throw new SAXException("Fatal Error encountered");
}  // end of method fatalError
}// end of class MyErrorHandler
```

```
/**
 * Class   : MyContentHandlerII
 * Purpose: implements the SAX interface and parses the document
 * based on the events that happen inside the document
 * it establishes the basic data structures that gather the
 * configuration information from the XML file.
 */
class MyContentHandlerII implements ContentHandler
{
  RouterRecord record;
  private ObjectInputStream input;
  public DemoGui gui = new DemoGui("SAAM XML SAXAgentParser  ver
          1.0   ");
  // Hold onto the locator for location information
  private Locator locator;
  // all global variables
  // temp variables to store values read from the XML file
  public String element = "" , value ="";  // element & value
  String tAgentName ="" , tNodeName ="", tParam ="";
  String [] []  agentArray = new String [100][15];
  // array of agent objects
  int agentArrayLength = 0;
  // index of agents in the agent array
  int paramCounter    = 4;
  // index of number of parameters per  agent

  /**
  * Method      : setDocumentLocator
  * Purpose     : This method gives the capability to define the
    exact location while
  * parsing the XML file
  * Parameters : Locator locator
  */
  public void setDocumentLocator(Locator locator)
  {
    System.out.println("    * setDocumentLocator() called");
    // We save this for later use if desired.
    this.locator = locator;
  }

  /**
   * Method      : startDocumnet
   * Purpose     : In this method there can be any kind of
        statements
   * that we would like to occur when we first open an XML
   * document
   * Parameters :
   * throws      : SAXException when things go wrong
  */
  public void startDocument() throws SAXException
  {
    System.out.println("Parsing begins...");
  }
```

```
/**
 * Method      : startElement
 * Purpose     : This reports the occurrence of an actual
   element.  It will include
 * he element's attributes, with the exception of XML vocabulary
 * specific attributes like "DTD",....
 * Parameters : String namespaceURI, String localName,
 *               String rawName, Attributes atts)
 * throws      : SAXException when things go wrong
*/
public void startElement(String namespaceURI, String
         localName,String rawName, Attributes atts)
  throws SAXException
{
  System.out.print(" \n startElement: " + localName);
  element = localName;
}



/**
 * Method      : characters
 * Purpose     : This method returns the real value stored in the
    XML element
 * It is then converted to a string " array of characters
 * Parameters : char[] ch, int start, int end
 * throws      : SAXException when things go wrong
*/
public void characters(char[] ch, int start, int end)throws
    SAXException
{
  String s = new String(ch, start, end);
  value = s;
}

/**
 * Method      : endElement
 * Purpose     : Indicates the end of an element is reached.
    Note that
 * the parser does not distinguish between empty elements and
 * non-empty elements, so this will occur uniformly.
 * We gather the value of the element when we reach the end
 * of the element
 * Parameters : String namespaceURI, String localName, String
 * rawName
 * throws      : SAXException when things go wrong
*/
public void endElement(String namespaceURI, String
        localName,String rawName)
  throws SAXException
{
  System.out.println("\n element = " + element + "   value =   "
  + value);
```

91

```java
    if (element.equals("NODE"))
    {
      gui.sendText("======== The beginning of a new node  =====");
    }
    else if   (element.equals("NodeName"))
    {
      tNodeName = value;
      gui.sendText(" Node Name : " + tNodeName);
    }
    else if (element.equals("Agent"))
    {
      gui.sendText("======== The beginning of an Agent   ======");
    }
    else if (element.equals("AgentName"))
    {
      tAgentName = value;
      gui.sendText(" AgentName : " + tAgentName);
    }
    else if (element.equals("Param"))
    {
      tParam = value;
      gui.sendText(" Param " + paramCounter +" : " + tParam);
    }
    if (localName.equals("AgentName"))
    {
      agentArray[agentArrayLength][3] = tAgentName; // agent name
    }
    else if (localName.equals("Param"))
    {
      agentArray[agentArrayLength][paramCounter] = tParam;
      paramCounter++;
    }
    else if (localName.equals("Agent"))
    {
      // at the end of Agent element, add an entry to the Agent
         array
      agentArray[agentArrayLength][0] = tNodeName; // agent name
      gui.sendText(" Add the agent to the agent array ... ");
      paramCounter = 4;
      agentArrayLength++;    // increase number of agents
    }
}  // end endElement method
```

```java
/**
 * Method      : endDocument
 * Purpose     : This method indicates the end of the XML
   document.
 * It is reached when it finishes all the parsing events inside
 * that
 * we would like to occur when we first open an XML document
 * Parameters : None
 * throws      : SAXException when things go wrong
 */
public void endDocument() throws SAXException
{
    System.out.println("...Parsing ends.");
    gui.sendText("=========================================");
    gui.sendText("Start creating interfaces when reaching the end
                    of the document");
    gui.sendText("==============================      =====");
    // start creating interfaces on the nodes
    try
    {
        input = new ObjectInputStream(
                        new FileInputStream( "RouterInfo.txt" ) );
    }
    catch ( IOException e )
    {
        gui.sendText(e.toString());
        gui.sendText("Error Opening File");
    }
    try
    {
        while (true)
        {
            record = ( RouterRecord ) input.readObject();

            for (int idx = 0; idx < agentArrayLength; idx++)
            {
            //   gui.sendText(" record NodeName: " + record.nodeName);
            //   gui.sendText(" record ipv4: " + record.ipv4Address);
                if ( agentArray[idx][0].equals(record.nodeName))
                {
                    agentArray[idx][1] = record.ipv4Address;
                    agentArray[idx][2] = record.v4Port;
                    gui.sendText(" IPV4 address : " + record.ipv4Address +
                            "  Emulation Port  : " + record..v4Port );
                }
            }
        }
    }
    catch ( EOFException eof )
    {
        gui.sendText(eof.toString());
    }
    catch ( IOException ioex )
```

```
    {
      gui.sendText(ioex.toString());
    }
    catch ( ClassNotFoundException cnf )
    {
      gui.sendText(cnf.toString());
    }
    gui.sendText("Display the agent array which has  " +
        agentArrayLength + "  agents");
    for (int idx = 0; idx < agentArrayLength; idx++)
    {
      gui.sendText(agentArray[idx][0]+ "/" + agentArray[idx][1] +
                "/" + agentArray[idx][2]+ "/" +
      agentArray[idx][3]+ "/" + agentArray[idx][4]);
    }
    // sending the agents to the nodes
    StringBuffer agentStringBuffer = new StringBuffer("");
    for (int agentCounter = 0; agentCounter < agentArrayLength;
        agentCounter++)
    {
      agentStringBuffer.append("saam.agent.applications.");    //
            the place where application agents are
      agentStringBuffer.append(agentArray[agentCounter][3]);
            // agent name
      DemoInitInfo.sendAgent(gui,agentArray[agentCounter],
      agentStringBuffer.toString());
      agentStringBuffer.delete(0,agentStringBuffer.length());
      // clear the string buffer to hold a new agent
    } // end for
    try
    {
      input.close();
    }
    catch(IOException oie)
    {
      gui.sendText(oie.toString());
    }
} // end of endDocument Method


/**
 * Method       : processingInstruction
 * Purpose      : This method is used when processing instructions
   are found
 * in the XML file. The current XML file doesn't support PI
 * Parameters : String target, String data
 * throws       : SAXException when things go wrong
*/
public void processingInstruction(String target, String
        data)throws SAXException
{
  System.out.println("PI: Target:" + target + " and Data:" +
    data);
```

94

```
}

/**
 * Method      : startPrefixMapping
 * Purpose     : This method is used in case XML name spaces are
   used.
 * In the current use of XML. name spaces are not supported.
 * Parameters : String prefix, String uri
 * throws      : SAXException when things go wrong
 */
public void startPrefixMapping(String prefix, String uri)
{
  System.out.println("Mapping starts for prefix " + prefix +
                            " mapped to URI " + uri);
}

/**
 * Method      : endPrefixMapping
 * Purpose     : This method is used in case XML name spaces are
     used.
 * In the current use of XML. name spaces are not supported.
 * Parameters : String prefix, String uri
 * throws      : SAXException when things go wrong
 */
public void endPrefixMapping(String prefix)
{
  System.out.println("Mapping ends for prefix " + prefix);
}

/**
 * Method      : characters
 * Purpose     : This method provides the ability to report white
   spaces
 * Parameters : char[] ch, int start, int end
 * throws      : SAXException when things go wrong
 */
public void ignorableWhitespace(char[] ch, int start, int
     end)throws SAXException
{
  String s = new String(ch, start, end);
  System.out.println("ignorableWhitespace: [" + s + "]");
}

/**
 * Method      : characters
 * Purpose     : This method will report an entity that is
   skipped by the parser.
 * This should only occur for non-validating parsers, and then
 * is still
 * implementation-dependent behavior.
 * Parameters : String name
 * throws      : SAXException when things go wrong
 */
```

```java
      public void skippedEntity(String name) throws SAXException
      {
        System.out.println("Skipping entity " + name);
      }
}
class RouterRecord implements Serializable
{
  public String nodeName;
  public String ipv4Address;
  public String v4Port;

  public RouterRecord()
  {
      this( "", "", "" );
  }

  /**
    * Constructor
    *
    * @param    nName    router's name
    * @param    ipv4     router's ipv4 address
    * @param    portNo   router's port number(MIPH)
    */
  public RouterRecord( String nName, String ipv4, String portNo )
  {
    nodeName = new String( nName );
    ipv4Address = new String (ipv4);
    v4Port = new String(portNo);
  }
}
```

APPENDIX D. DEMOINITINFO CLASS SOURCE CODE

```
//  Jan01[Fatih] - sendAgent() method has been chanced.
//20Oct00[Colwell] - Multiple Instance Per Host code added
//11May00[Xie] - added sendAgent() and sendMessage()
//10May00[Brock] -created (initHosts is a modification of the old
initRouter/initServer).

package saam.demo;

import saam.control.*;
import saam.message.*;
import saam.net.*;
import saam.util.*;
import java.net.*;
import java.io.*;
import java.util.Vector;
import java.util.Enumeration;


/**
 * This class is used to hold the data neccessary to configure
 * the information for one host. Once configured, the host will
 * know the addresses for itself(IPv4), its interfaces(IPv6 and MAC),
 * and the interfaces for its neighbors.  Furthermore, its ARP cache,
 * and emulation table (mapping IPv6 addresses to IPv4 addresses)
 * will be initialized.
 */
public class DemoInitInfo
{
  private final static int defaultEmulationPort = 9002;
 //SAAM UDP emulation port (IPv4 world)

  private Vector interfaces = new Vector();
  private Vector emTable     = new Vector();
  private Vector arpCache   = new Vector();

  private Vector agents = new Vector();

  private DemoGui gui;
  private InetAddress myIPv4Addr;
  private int timeScale;
  private Configuration serverConfig;
  private EmulationPort emPort;
 // DemoStation contacts player instance on this -crc
  private String iPv4AddrTag;
 // Used to dynamically search for mulit-instances per host -crc
  private String nodeName;
 // Sent to each player instance on every host -crc

  // contains every interface and its associated IPv4 address
```

97

```java
private static Vector interfaceMapping = new Vector();

/**
  * DemoInitInfo constructor called by routers
 * @param gui the window to output status messages to
 * @param myIPv4Addr the IP ver 4 address of the host
 * @param emPort IPv4 TCP port DemoStation uses to initialize a
 * translator.
 * @param timeScale the scaling factor for the server.
 * @param nodeName String representing the player identifier displayed
 * in the player's mainGui frame
  */
public DemoInitInfo( DemoGui gui, InetAddress myIPv4Addr,
                     EmulationPort emPort, // add port -crc
    int timeScale, String nodeName )
{
  this.gui = gui;
  this.myIPv4Addr = myIPv4Addr;
  iPv4AddrTag = myIPv4Addr.toString(); // -crc
  this.timeScale = timeScale;
  this.emPort = emPort; // crc
  this.nodeName = nodeName; //crc
  addCoreAgents();
}

/**
 * DemoInitInfo constructor called by servers.
 * @param gui the window to output status messages to
 * @param myIPv4Addr the IP ver 4 address of the host
 * @param emPort IPv4 TCP port DemoStation uses to initialize a
 * translator.
 * @param timeScale the scaling factor for the server
 * @param nodeName String representing the player identifier displayed
 * in the player's mainGui frame
 * @param serverConfig the configuration info for the server
 */
public DemoInitInfo( DemoGui gui, InetAddress myIPv4Addr,
            EmulationPort emPort, // add port -crc
      int timeScale, String nodeName, Configuration serverConfig )
{
  this.gui = gui;
  this.myIPv4Addr = myIPv4Addr;
  iPv4AddrTag = myIPv4Addr.toString(); // -crc
  this.serverConfig = serverConfig;
  this.timeScale = timeScale;
  this.emPort = emPort; // crc
  this.nodeName = nodeName; //crc
  addCoreAgents();
}
/**
 * Add the core agents to the agent list,  This must be called
 * prior to adding any additional agents.
 */
```

```java
private void addCoreAgents(){
  // define the core agents
  agents.add("saam.agent.router.Scheduler");
  agents.add("saam.agent.router.ARPCache");
  agents.add("saam.agent.router.FlowRoutingTable");
}


/**
 * Method used to query the host IPv4 address this object resides on.
 */
public String queryIPv4AddrTag()
{ //-crc
  return(iPv4AddrTag);
}


/**
 * Returns the name of this node
 */
public String getNodeName()
{
 // Added so each instance can be manually identified -crc
  return(nodeName);
}


/**
 * Add an agent to this host
 * @param agent the additional agent to send to this host
 */
public void addAgent( String agent )
{
    agents.add( agent );
}


/**
 * Initializes a new interface for this host, assigning it a unique
 * MAC address.  The interface is added to this host's interface list.
 * @param IPv6String the IP ver 6 address of the interface.
 * @return an <code>InterfaceID</code> representing the new interface.
 */
public InterfaceID getNewInterface( String IPv6String, int bandwidth,
    int maskBits )
{
  try
  {
    IPv6Address address = new IPv6Address(
        IPv6Address.getByName(IPv6String).getAddress() );
    byte macAddress = (byte) interfaceMapping.size();
    InterfaceID interfaceID = new InterfaceID( address, macAddress,
        bandwidth, (byte)maskBits);
    interfaces.add( interfaceID );
    interfaceMapping.add( new Node( myIPv4Addr, emPort, interfaceID
            )); //add emPort -crc
    return interfaceID;
```

```
    }
    catch( UnknownHostException uhe )
    {
      gui.sendText( uhe.toString() );
      return null;
    }
  }


/**
 * Lets this host know that it is connected to another host
 * via that host's interface.  Adds an ARP entry and emulation
 * table entry for this host.
 * @param interfaceID the interface on the other host.
 */
public void isConnectedTo( InterfaceID interfaceID)
{
  // find the correct mapping info
  Enumeration iter = interfaceMapping.elements();
  while (iter.hasMoreElements())
  {
    Node curr = (Node)iter.nextElement();
    if (curr.getNextInterface() == interfaceID)
    // use saam port here.
    {
      int saamPort = curr.getEmulationPort().toInt() - 1;
      //always one less than emPort -crc
      emTable.add(new EmulationTableEntry( interfaceID.getIPv6(),
          curr.getIPv4address(),
      new EmulationPort(saamPort)   ));
      // use corresponding saam port -crc
      arpCache.add( new ARPCacheEntry(interfaceID.getIPv6(),
          interfaceID.getMAC()));
      break;
    }
  }
}


/**
 * Activates this host, sending the information to the
 * translator running on the proper machine.
 * Inputs info vectors specifying interface, EmulationTable, arpCache,
 * IPv4 address of target machine, and TimeScale factor
 * objects for a router.  Uses them to build a SAAM packet, along
 * with core and serverAgent code. Then transmits
 * the packet to target machine using the SAAM emulation TCP port.
 */
public void activate(DemoGui gui)
{
  //  Oct00[Colwell] -Multi Instance Per Host mods
  // 11May00[Brock] -Pulled out the packet sending code in to
     sendPacket
  // 26Apr2000[Colwell] -Added ObjectOutputStream code to match
     Translator mods
```

```java
// 14Dec99[Akkoc]  -Modified for my test topology and new
   emulationtable format, hellos disabled
//01Aug99[Vrable] -Created
PacketFactory packet = new PacketFactory();
gui.sendText("Destination IPv4:\n\t" + myIPv4Addr + "\n\tPort = " +
   emPort);
// time scale message must be sent before any other messages
packet.append(new TimeScale(timeScale));
// demoHello must be sent before any agents
DemoHello helloMessage = new DemoHello(interfaces, getNodeName());
//MIPH mod, pass node name -crc
packet.append(helloMessage);
if ( serverConfig != null )
{
   agents.add("saam.agent.server.ServerAgent");
}
try
{
   //now append agents...
   Enumeration agentIter = agents.elements();
   while (agentIter.hasMoreElements())
   {
     String agent = (String) agentIter.nextElement();
     packet.append( agent , (byte) 0);
   }
   if ( serverConfig != null )
   {
   // add configuration information, must be sent after server agent
     packet.append(serverConfig);
   }
}
catch (IOException ioe)
{
   gui.sendText(ioe.toString());
   gui.sendText("problem in agents");
}

// display list of agents being sent
gui.sendText("\n\tLoading agents:");
Enumeration agentIter = agents.elements();
while (agentIter.hasMoreElements())
{
   String agent = (String) agentIter.nextElement();
   gui.sendText( "\t\t" + agent );
}
//add entries to the EmulationTable
Enumeration el = emTable.elements();
//gui.sendText("EmulationTable:");  //debugging  -crc
while (el.hasMoreElements())
{
//packet.append( (EmulationTableEntry)( el.nextElement()));
   EmulationTableEntry ete = (EmulationTableEntry)(
      el.nextElement());
```

101

```
      /*try
      {
        if
        (ete.getIPv4Address().toString().equals(myIPv4Addr.toString()))
        {
          gui.sendText("Emulation 1   --->>" +
              ete.getIPv4Address().toString());
          ete.setIPv4Address(InetAddress.getByName("127.0.0.1"));
          gui.sendText("Emulation 1.1   --->>" +
              ete.getIPv4Address().toString());
        }
      }
      catch(UnknownHostException ue)
      {
        gui.sendText(ue.toString());
      }*/
      packet.append( ete );
    //gui.sendText("\t" + ete); //debugging
  } //end of while

  //add entries to the ARPCache
  Enumeration e2 = arpCache.elements();
  while (e2.hasMoreElements())
  {
    packet.append((ARPCacheEntry) e2.nextElement());
  } //end of while
  sendPacket( gui, myIPv4Addr, emPort.toInt(), packet );
    // add emPort arg -crc
  gui.sendText("\n");
}//end activate()


/**
 * Sends a packet.
 * @param gui location to send status messages.
 * @param destIPv4 destination IP ver 4 address.
 * @param packet the packet to send.
 */
private static void sendPacket(DemoGui gui, InetAddress destIPv4, int
        v4Port, //add port - crc
          PacketFactory packet)
{
  byte[] packetArray = packet.getBytes();
  //Note:  getBytes also sets packet object up to be reused for a new
            message
  try
  {
    gui.sendText("sendPkt:pre socket: v4@: "+destIPv4.toString()+ "
        port: "+v4Port);
    Socket socket = new Socket(destIPv4, v4Port);   // add port -crc
    socket.setTcpNoDelay(true);
    OutputStream os = socket.getOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(os);
```

102

```java
      //new stuff for revised translator-crc
      try
      {
        oos.writeObject(packetArray);
      }
      catch (IOException e)
      {
        e.printStackTrace();
      }
      oos.flush();
      oos.close();
      os.close();
      socket.close();
      gui.sendText("Packet sent.  Length = " + packetArray.length);
    }
    catch (Exception e)
    {
      System.out.println(e.toString());
      System.out.println("-- Problem in DemoInitInfo.sendPacket");
    }
  }//end sendPacket()


  /**
   * This method takes the agent name and paramaters belong to that gent
   * creates a Saam packet , appends agent and paramaters to the packet
   * and pass it to sendPacket method to be able to send to the router
   * which this agent will be installed
   * @param    inGui
   * @param    agentArray  string array holds the agent parameters
   * @param    agentName    name of the agent
   */
  public static void sendAgent(DemoGui inGui, String [] agentArray,
      String agentName)
  {
    PacketFactory  packet = new PacketFactory();
    InetAddress destIPv4 = null;
    // port number for MIPH
    int v4Port = Integer.parseInt(agentArray[2]);
    int paramCount =4;
    byte type = 0;
    try
    {
      destIPv4 = InetAddress.getByName(agentArray[1]);
    }
    catch (UnknownHostException uhe)
    {
      inGui.sendText(uhe.toString());
    }
    // check if this agent is traffic agent or some other agent
    // traffic agent type should be 30
    // this is needed when the packet is striped by the PacketFactory.
    if (agentArray[4] == null)
```

103

```
      {
        type = 0;
      }
      else
      {
        type = 30;
      }
      try
      {
        packet.append(agentName, type); // append agent
        while(agentArray[paramCount] != null)
        {
          packet.appendInfo(agentArray[paramCount]);
          // append agent parameters
          paramCount++;
        }
      }
      catch (IOException ioe)
      {
        inGui.sendText(ioe.toString());
        inGui.sendText("Problem in appending agent to packet");
      }

      sendPacket(inGui, destIPv4, v4Port, packet);
      inGui.sendText("Successfully sent agent <" + agentName + "> to:   " +
          agentArray[1] );
  }//end sendAgent()


/**
 * Sends a message.
 * @param gui location to send status messages.
 * @param destIPv4 destination IP ver 4 address.
 * @param message to send.
**/
public static void sendMessage(DemoGui inGui, String hostName, int
      v4Port, //-crc
        Message msg)
{
  PacketFactory  packet = new PacketFactory();
  InetAddress destIPv4 = null;
  try
  {
    destIPv4 = InetAddress.getByName(hostName);
    //return; //-comment out -crc
  }
  catch (UnknownHostException uhe)
  {
    inGui.sendText(uhe.toString());
  }
  packet.append(msg);
  sendPacket(inGui, destIPv4, v4Port, packet);  //add emPort -crc
  inGui.sendText("Successfully sent message <" + msg + "> to:   " +
```

104

```
          hostName +
        " at IPv4 Port: " + v4Port);
   }//end sendMessage()


   public EmulationPort queryEmulationPort()
   { // -crc
     return(emPort);
   }
}//end DemoInitInfo class


/**
 * This internal class is used to hold the information mapping
 * all IPv6 interface to IPv4 as the host information is being
configured.
 */
class Node
{
   InetAddress IPv4address;
   InterfaceID nextInterface;
   EmulationPort v4Port;   //crc

   public Node(InetAddress IPv4address, EmulationPort v4Port, InterfaceID
        nextInterface)
   { //add port arg-crc
     this.IPv4address = IPv4address;
     this.nextInterface = nextInterface;
     this.v4Port = v4Port; //crc
   }

   public InetAddress getIPv4address()
   {
     return(IPv4address);
   }

   public InterfaceID getNextInterface()
   {
     return(nextInterface);
   }

   public EmulationPort getEmulationPort() // added -crc
   {
     return(v4Port);
   }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. PACKETFACTORY CLASS SOURCE CODE

```java
/**
  * This method can be used to append a ResidentAgent by name to an
  * outgoing SAAMPacket.  To later retrieve the entire packet
  * (with header) as a byte array, call the getBytes method.
  * @param residentAgentClassName The String name of the ResidentAgent
  * classfile to be appended.
  */
 public void append(String residentAgentClassName, byte type)
   throws IOException
{
  if (bytesRetrieved)
  {
    packet = null;
    numberOfOutputMsgs = 0;
    bytesRetrieved = false;
  }
  String name = residentAgentClassName;
   //*****Mod fixes "finding the agent problem"-  Get current
    execution dir as base.
   //  Use that reference to build absolute path/filename for the
    agent.. -crc
  String executionDir =  FileIO.getWorkingDir();
  //Pull agent code from the proper build.
  System.out.println("PacketFactory: Property Current Dir: "+
     executionDir);
  File tempFile = new File(executionDir);
  System.out.println("PacketFactory: Property file:
     "+tempFile.getName() );
  String lookFor = tempFile.getName();
  if ( lookFor.equalsIgnoreCase("SAAM"))
  {
    System.out.println("PacketFactory: executed inside the saam
        package"); //remove "saam"
     executionDir = new StringBuffer(executionDir+File.separatorChar).
        substring( 0, executionDir.length() -
        saam".length()).toString();
  }
  else
  {
     System.out.println("PacketFactory: exececuted outside saam
        package");
  }
  String fileName = executionDir +
    residentAgentClassName.replace('.', File.separatorChar) +
    ".class";

/****  Above code replaces the following long-time troublesome business
//   String fileName = "C:\\WINNT\\Profiles\\administrator\\Desktop\\
//   Java\\saamjuly\\saamxpand1" + File.separatorChar +
```

```java
//   String fileName = "C:\\hakkoc"+File.separatorChar +
//   String fileName = ".." + File.separatorChar +   //for KAWA
// String fileName = "..\\.." + File.separatorChar +     //for Jbuilder
// residentAgentClassName.replace('.', File.separatorChar); fileName +=
// ".class"; ****/

//     gui.sendText("File name: " + fileName);
    System.out.println("PacketFactory: File name: " + fileName);
    FileInputStream fis = null;
    try
    {
      fis = new FileInputStream(fileName);
    }
    catch (IOException ioe)
    {
      throw new IOException(
        "Problem reading ResidentAgent: " + fileName);
    }
    byte nameLength = (byte) name.getBytes().length;
    byte[] byteCode = new byte[fis.available()];
    short length = (short) fis.read(byteCode);
    packet = Array.concat(packet, type);
    packet = Array.concat(packet, nameLength);
    packet = Array.concat(packet, name.getBytes());
    packet = Array.concat(packet,
    PrimitiveConversions.getBytes(length));
    packet = Array.concat(packet, byteCode);
    numberOfOutputMsgs++;

//     gui.sendText("--appended ResidentAgent:" +
    System.out.println("--appended ResidentAgent:" +
      "\n   type:              " + type +
      "\n   name:              " + name +
      "\n   byteCode length:   " + length +
      "\n   # of messages:     " + numberOfOutputMsgs +
      "\n   packet length:     " + packet.length);
  }

/**
   * This method can be used to append an Agent information to an
   * outgoing SAAMPacket.  To later retrieve the entire packet
   * (with header) as a byte array, call the getBytes method.
   * @param informationAgent The String name of the agent information.
   * [fatih] dec00
   */
  public void appendInfo(String informationAgent) throws IOException
  {
    if (bytesRetrieved)
    {
      packet = null;
      numberOfOutputMsgs = 0;
      bytesRetrieved = false;
    }
```

108

```java
      String info = informationAgent;

      byte infoLength = (byte) info.getBytes().length;
      packet = Array.concat(packet, infoLength);
      packet = Array.concat(packet, info.getBytes());
   } // end appendInfo


   /**
    * This method is used to extract the individual Class
    * Objects that are represented in the packet.  These Class
    * Objects are either of type 0 (ResidentAgent) or 1 (Message).<p>
    * If a ResidentAgent is received, a Class Object is created
    * that represents the agent.  That Class Object is then sent to
    * the ControlExecutive for screening and agent instantiation.<p>
    * If a Message is received, that Message is instantiated and sent
    * to the ControlExecutive for further processing.
    */
   private void processPacket(byte[] packet)
   {
      //see saam.util for PrimitiveConversions and Array classes
      long timeStamp = PrimitiveConversions.getLong(
                                  Array.getSubArray(packet, 0, 8));
      byte numberOfInputMsgs = packet[8];

      gui.sendText("\nProcess packet: " +
        "\n  size:          " + packet.length +
        "\n  # of Messages: " + numberOfInputMsgs +
        "\n  timeStamp:     " + timeStamp);

      //now we trim the packet by removing the header.
      packet = Array.getSubArray(packet, 9, packet.length);
      //used to track the current position in the array.
      int index = 0;
      short length = 0;
      String elementName = "";
      Class message = null;
      //extract and process each atomic element of the packet
      //separately.  Here we assume the packet is a properly
      //formatted SAAMPacket when it arrives, and that the
      //length is less than the max allowed.
      for(int i = 1; i <= numberOfInputMsgs; i++){
        gui.sendText("\n  Processing Element["+i+"]:");
        byte type = packet[index++];
        gui.sendText("    type:  " + type);
        byte[] bytes;
        switch(type)
        {
          case Message.RESIDENT_AGENT:
          case Message.MESSAGE_DEFAULT_TYPE:
          case Message.INTERFACE_SHUTDOWN:
            //retrieve the number of bytes the class name occupies
           byte nameLength = packet[index++];
```

109

```
    //extract the name of the class file as a byte array
     byte[] elementNameArray = Array.getSubArray(packet, index,
    index + nameLength);
    index += nameLength;
    //convert the name back into a String
    elementName = new String(elementNameArray);
     gui.sendText("    Name: " + elementName);
   length = PrimitiveConversions.getShort(Array.getSubArray(packet,
        index,index + 2));
    index += 2;
    gui.sendText("    Length:        " + length);
    bytes = Array.getSubArray(packet, index, index + length);
    index += length;
    if (type == Message.RESIDENT_AGENT)
    {
       gui.sendText("    This is a ResidentAgent");
       //Assume this class is of type ResidentAgent
       try
       {
       //Attempt to define the class using the current
       //class loader.
         loader.defClass(elementName, bytes);
       }
       catch (LinkageError le)
       {
        //If the loader already has a definition for the class
        //a LinkageError will be thrown.  If this happens, we
        //need to instantiate a new class loader and use it to
        //define the class.  A nice little trick we learned from
        //page 55 of Jason Hunter's "Java Servlet Programming" book.
         gui.sendText(le.toString());
         gui.sendText("Class was previously loaded...");
         gui.sendText("Replacing old ClassLoader...");
         Loader newLoader = new Loader();
         newLoader.defClass(elementName, bytes);
       }
       try
       {
        //message is of type Class.
         message = Class.forName(elementName, true, loader);
       }
       catch (ClassNotFoundException cnfe)
       {
         gui.sendText(cnfe.toString());
       }
       gui.sendText(message.toString());
       ResidentAgentEvent rae = new ResidentAgentEvent(
        this.toString(),this,
        ControlExecutive.SAAM_CONTROL_PORT,message);
        Try
        {
          gui.sendText("    Forwarding on channel " +
            ControlExecutive.SAAM_CONTROL_PORT);
```

110

```java
          controlExec.talk(rae);
    }
    catch (ChannelException tde)
    {
      gui.sendText(tde.toString());
    }
}//end ResidentAgent
else if (type == Message.MESSAGE_DEFAULT_TYPE){
  gui.sendText("    This is a Message");
  //Assume this class is of type Message.
  try{
    //message is of type Class.
    message = Class.forName(elementName);
  }
  catch(ClassNotFoundException cnfe){
    gui.sendText("Bytecode for: " + elementName + " not
        found.");
  }
  try{
    //Call the constructor from within this Class that
    //takes a byte array as its only argument
    Constructor cons = message.getConstructor(new Class[]
        {byte[].class});
    //Create the instance of this Message
    Message instance =
      (Message) cons.newInstance(new Object[] {bytes});
    gui.sendText(instance.toString());
    MessageEvent me = new MessageEvent(
        this.toString(),
        this,
        ControlExecutive.SAAM_CONTROL_PORT,
        instance);

    //send this MessageEvent on the Control port.
    try{
      gui.sendText("    Forwarding on channel " +
      ControlExecutive.SAAM_CONTROL_PORT);
      controlExec.talk(me);
    }
    catch(ChannelException tde){
      gui.sendText(tde.toString());
    }
  }
  catch(Exception e){
    //need to notify sender that we have no classfile
    //with this name
    gui.sendText(e.toString());
  }//try-catch
}//end else if default message type '1'
//Hasan UYSAL
else if (type == Message.INTERFACE_SHUTDOWN){
  gui.sendText("    This is an Interface Shutdown message.");
```

111

```java
        InterfaceShutdown failure = new InterfaceShutdown(bytes);
        MessageEvent failMes = new MessageEvent(
            this.toString(),
            this,
            ControlExecutive.SAAM_CONTROL_PORT,
            failure);

        try{
          controlExec.talk(failMes);
        }
        catch(Exception ex){
          gui.sendText("Problem with interface shutdown message.");
          continue;
        }
      }
    break;


//THIS CASE PROCESSES THE HEARBEATQUERY MESSAGE
//added by Efraim Kati
case Message.HEARTBEAT_QUERY:

  //retrieve the bytecode of the Object
  bytes = Array.getSubArray(packet, 1, packet.length);
  gui.sendText("    This is a  HeartbeatQuery message");

  //Create the instance of this Message
  HeartbeatQuery hbq = new HeartbeatQuery(bytes);
  gui.sendText(hbq.toString());

  MessageEvent hbqMe = new MessageEvent(
            this.toString(),this,
            ControlExecutive.SAAM_CONTROL_PORT,hbq);

  //send this MessageEvent on the Control port.
  try{
    gui.sendText("    Forwarding on channel " +
    ControlExecutive.SAAM_CONTROL_PORT);
    controlExec.talk(hbqMe);
  }
  catch(ChannelException tde){
    gui.sendText(tde.toString());
  }
  break;

//THIS CASE PROCESSES THE HEARBEATRESPONSE MESSAGE
//added by Efraim Kati
case Message.HEARTBEAT_RESPONSE:

  //retrieve the bytecode of the Object
  bytes = Array.getSubArray(packet, 1, packet.length);
  gui.sendText("    This is a  HeartbeatResponse message");
```

112

```java
        //Create the instance of this Message
        HeartbeatResponse hbr = new HeartbeatResponse(bytes);
        gui.sendText(hbr.toString());

        MessageEvent hbrMe = new MessageEvent(
                           this.toString(),this,
                           ControlExecutive.SAAM_CONTROL_PORT, hbr);

        //send this MessageEvent on the Control port.
        try{
          gui.sendText("      Forwarding on channel " +
          ControlExecutive.SAAM_CONTROL_PORT);
          controlExec.talk(hbrMe);
        }
        catch(ChannelException tde){
          gui.sendText(tde.toString());
        }
        break;

    //Henry
    case Message.FLOW_REQUEST:
    case Message.FLOW_RESPONSE:
    case Message.RESOURCE_ALLOCATION:
    case Message.SLS_TABLE_ENTRY:

        //retrieve the length of the Object
        length = PrimitiveConversions.getShort
                  (Array.getSubArray(packet, index,index + 2));
        index += 2;

//          gui.sendText("      Length:        " + length);
        bytes = Array.getSubArray(packet,index,index + length);
        index += length;

        if(type == Message.FLOW_REQUEST){
          gui.sendText("      This is a FlowRequest Message");
          controlExec.processMessage(bytes, "FlowRequest");
        }//end flow request

        else if(type == Message.FLOW_RESPONSE){
          gui.sendText("      This is a FlowResponse Message");
          controlExec.processMessage(bytes, "FlowResponse");
        }//end flow response

        else if(type == Message.RESOURCE_ALLOCATION){
          gui.sendText("      This is a ResourceAllocation Message");
          controlExec.processMessage(bytes, "ResourceAllocation");
        }//end resource allocation

        else if(type == Message.SLS_TABLE_ENTRY){
          gui.sendText("      This is a SLSTableEntry Message");
          if (bytes.length == SLSTableEntry.REMOVE_SLS_TYPE) {
            controlExec.processMessage(bytes, "SLSTableEntry");
```

113

```java
        }
        else {
          processMessage(bytes, this.toString(), "SLSTableEntry");
        }
      }//end slstableentry
      break;

    case Message.FLOW_TERMINATION:
      length = 4;

//    gui.sendText("  Length:        " + length);
      bytes = Array.getSubArray(packet,index,index + length);
      index += length;
      gui.sendText("    This is a FlowTermination Message");
      controlExec.processMessage(bytes, "FlowTermination");
      break;

    //Hasan AKKOC
    case Message.DCM:
    case Message.PARENT_NOTIFICATION:

      if(type == Message.DCM){
        gui.sendText("    This is a DCM  Message");

        try{
          DCM  dcm = new DCM(packet);
//          gui.sendText("DCM message ia created.");
          gui.sendText(dcm.toString());
          MessageEvent me = new MessageEvent(
                    this.toString(), this,
                    ControlExecutive.SAAM_CONTROL_PORT, dcm);

          //send this MessageEvent on the Control port.
          try{
            gui.sendText("    Forwarding on channel " +
            ControlExecutive.SAAM_CONTROL_PORT);
            controlExec.talk(me);
//            gui.sendText("DCM is sent to ControlExecutive.");
          }
          catch(ChannelException tde){
            gui.sendText(tde.toString());
          }
        }
        catch(Exception e){
          gui.sendText(e.toString());
        }//try-catch
      }//DCM

      else if(type == Message.PARENT_NOTIFICATION){
        gui.sendText("    This is a ParentNotification Message");
        //Assume this class is of type Message.
        try{
          ParentNotification pn = new ParentNotification(packet);
```

114

```java
        MessageEvent me = new MessageEvent(
                this.toString(), this,
              ControlExecutive.SAAM_CONTROL_PORT, pn);

        //send this MessageEvent on the Control port.
        try{
          gui.sendText("    Forwarding on channel " +
          ControlExecutive.SAAM_CONTROL_PORT);
          controlExec.talk(me);
        }
        catch(ChannelException tde){
          gui.sendText(tde.toString());
        }
      }
    catch(Exception e){
      gui.sendText(e.toString());
    }//try-catch
  }
  break;

case Message.UCM:
  gui.sendText("    This is a UCM  Message");
  int flowIdOfServer = PrimitiveConversions.getInt
      (Array.getSubArray(packet,index, index + 4)) - 1;
        gui.sendText("    Flow id of the sever is " +
        flowIdOfServer);
  if (!controlExec.isServer()){
    //I need UCM and the LSAs together
    //need to pass the value of numberOfInputMsgs to UCM handler
    gui.sendText("    This is a router so forwarding the packet
                    to ACE");
    packet = Array.concat(numberOfInputMsgs, packet);
    i = numberOfInputMsgs;   //cause for loop to break? [Xie]
    //create a ProtocolStackEvent and send it to ACE
    int channelToAce = ProtocolStackEvent.
          FROM_PACKETFACTORY_TO_ACE;
     ProtocolStackEvent stackEvent = new ProtocolStackEvent(
        this.toString(),
        this,
        channelToAce,
        packet);

    try{
      gui.sendText("    Forwarding on channel " + channelToAce);
      controlExec.talk(stackEvent);
    }
    catch(ChannelException tde){
      gui.sendText(tde.toString());
    }
  }
  else{
    //this is the server that UCM is destined
    int numberOfRoutersInUCM =
```

```
                    PrimitiveConversions.getInt(Array.getSubArray(
    packet, index + 20, index + 24));
  gui.sendText("    Number of reachable routers is " +
                  numberOfRoutersInUCM);

  int UCMLength = 4 + 16 + 4 + 4 +
                  (numberOfRoutersInUCM * IPv6Address.length);
  bytes = Array.getSubArray(packet, index, index + UCMLength);
  bytes = Array.concat(type, bytes);
  index += UCMLength;
  try{
    UCM ucm  =  new UCM(bytes);
    gui.sendText("\n    " + ucm.toString());

    MessageEvent me = new MessageEvent(
        this.toString(),
        this,
        ControlExecutive.SAAM_CONTROL_PORT,
        ucm);

    gui.sendText("    Forwarding packet on channel
    ControlExecutive.SAAM_CONTROL_PORT);
    controlExec.talk(me);
  }
  catch(Exception ex){
    gui.sendText(ex.toString());
  }
}//end else
break;

//Huseyin UYSAL
case Message.LSA:
  //I got a link state advertisement so I need to process
    accordingly
  //processLSAMessage();
  gui.sendText("    This is a Link State Advertisement");
  //Assume this class is of type Message.

  short messageLength =
      PrimitiveConversions.getShort(Array.getSubArray(packet,
      index, index + 2));

  LinkStateAdvertisement lsa = new  LinkStateAdvertisement
    (Array.getSubArray(packet, index - 1, index - 1 +
      messageLength));//Modified by Kuo
  index += messageLength - 1;

  //create a Protocol Stack event and send this to control exec
  MessageEvent me = new MessageEvent(
                          this.toString(),
                          this,
                          ControlExecutive.SAAM_CONTROL_PORT,
                          lsa);
```

116

```java
    try{
      gui.sendText("     Forwarding on channel " +
      ControlExecutive.SAAM_CONTROL_PORT);
      controlExec.talk(me);
    }
    catch(Exception e){
      gui.sendText(e.toString());
    }
    break;

// this case statement is added by altinkaya in March 2000
// as a requirement for the server probing process
case Message.PREVIOUS_NODE_ACT:
    gui.sendText("     This is a Previous Node Activation
      Message");
    PreviousNodeAct pna = new PreviousNodeAct();
    byte typeOfProbing;
    int probeID;
    IPv6Header v6Header = null;
    short payloadLength;

    typeOfProbing = packet[index++];
    probeID =
        PrimitiveConversions.getInt(Array.getSubArray(packet,
        index, index = index + 4));
    try{
      v6Header = new IPv6Header(Array.getSubArray(packet, index,
              index = index + 40));
    }
    catch (UnknownHostException uhe){
      gui.sendText("An exception occured while trying to read
          ipv6Header ");
    }//end try-catch

    payloadLength =
      PrimitiveConversions.getShort(Array.getSubArray(packet,
        index, index = index + 2 ));
    pna.setPacketFormat( typeOfProbing, probeID, v6Header,
        payloadLength );
    gui.sendText("     Type of probing " +
        pna.getTypeOfProbing(typeOfProbing));
    gui.sendText("     Probing idendification " + probeID );
     MessageEvent mep = new MessageEvent(
                          this.toString(),
                          this,
                          ControlExecutive.SAAM_CONTROL_PORT,
                          pna);      //me->mep -crc
    try{
      gui.sendText( "     Forwarding on channel " +
      ControlExecutive.SAAM_CONTROL_PORT);
      controlExec.talk(mep);     // me->mep -crc
    }
    catch( Exception e){
```

117

```java
            gui.sendText( e.toString());
        }
     break;

// this case statement is added by altinkaya in March 2000
   // as a requirement for the server probing process
   case Message.NEXT_NODE_ACT:
     gui.sendText("    This is a Next Node Activation Message");
     NextNodeAct nna = new NextNodeAct();
     byte typeOfProbingNNA;
     int probeIDNNA;
     IPv6Header v6HeaderNNA = null;
     short measurementIntervalNNA;
     byte nicID;

     typeOfProbingNNA = packet[index++];
     probeIDNNA =
       PrimitiveConversions.getInt(Array.getSubArray(packet, index,
         index = index + 4));
     try{
        v6HeaderNNA = new IPv6Header(Array.getSubArray(packet,
          index, index = index + 40 ));
     }
     catch (UnknownHostException uhe){
        gui.sendText("An exception occured while trying to read
          ipv6Header ");
     }//end try-catch
     measurementIntervalNNA = PrimitiveConversions.getShort(
          Array.getSubArray(packet, index, index = index + 2 ));
     nicID = packet[index++];

     nna.setPacketFormat(typeOfProbingNNA, probeIDNNA, v6HeaderNNA,
                          measurementIntervalNNA, nicID );

     gui.sendText("    Type of probing " +
             nna.getTypeOfProbing(typeOfProbingNNA));
     gui.sendText("    Probing idendification " + probeIDNNA);
     MessageEvent meNNA = new MessageEvent(
                                 this.toString(),
                                 this,
       ControlExecutive.SAAM_CONTROL_PORT, nna);
     try{
        gui.sendText( "    Forwarding on channel " +
        ControlExecutive.SAAM_CONTROL_PORT);
        controlExec.talk(meNNA);
     }
     catch (Exception e){
        gui.sendText( e.toString());
     }//end try-catch
   break;


// this case statement is added by altinkaya in March 2000
```

118

```java
      // as a requirement for the server probing process
      case Message.PROBE_RESULT:
        gui.sendText("    This is a Probe Result Message ");
        ProbeResult pr = new ProbeResult();
        int probeIDPR;
        byte numberResultsPR;
        byte typeofResultPR;
        short measurementResultPR;

        probeIDPR =
            PrimitiveConversions.getInt(Array.getSubArray(packet,
                index, index = index + 4));
        numberResultsPR = packet[index++];
        typeofResultPR = packet[index++];
        measurementResultPR = PrimitiveConversions.getShort(
            Array.getSubArray(packet, index, index = index + 2 ));
        pr.setPacketFormat(probeIDPR, numberResultsPR, typeofResultPR,
                           measurementResultPR);
        gui.sendText("Information about the Probe Result " +
                    pr.toString());
        MessageEvent mePR = new MessageEvent(
                                   this.toString(),
                                   this,
                                   ControlExecutive.SAAM_CONTROL_PORT,
                                   pr);
      try{
          gui.sendText("    Forwarding on channel " +
          ControlExecutive.SAAM_CONTROL_PORT);
          controlExec.talk(mePR);
      }
      catch (Exception e){
          gui.sendText(e.toString());
      }//end try-catch
break;


      // this case statement is added by FATIH in DEC 2000
      case Message.TRAFFIC_GENERATORS:

        String param = "";
        // this array keeps the ipv6 addresses of the nodes
        String [] agentInfoArray = new String [15];
        int agentInfoCounter = 0;

        gui.sendText("    This is a traffic generator agent ");
        //retrieve the number of bytes the class name occupies
        byte nameLengthThis = packet[index++];

        //extract the name of the class file as a byte array
        byte[] elementNameArrayThis = Array.getSubArray(packet, index,
        index + nameLengthThis);
        index += nameLengthThis;
```

119

```java
//convert the name back into a String
elementName = new String(elementNameArrayThis);
gui.sendText("    Name: " + elementName);

//retrieve the length of the Object
length =
  PrimitiveConversions.getShort(Array.getSubArray(packet,
      index, index + 2));
index += 2;
gui.sendText("    Length:         " + length);
bytes = Array.getSubArray(packet, index, index + length);
index += length;

//Assume this class is of type ResidentAgent
try
{
  //Attempt to define the class using the current
  //class loader.
  loader.defClass(elementName, bytes);
}
catch (LinkageError le)
{
  //If the loader already has a definition for the class
  //a LinkageError will be thrown.  If this happens, we
  //need to instantiate a new class loader and use it to
  //define the class.  A nice little trick we learned from
  //page 55 of Jason Hunter's "Java Servlet Programming" book.

  gui.sendText(le.toString());
  gui.sendText("Class was previously loaded...");
  gui.sendText("Replacing old ClassLoader...");
  Loader newLoader = new Loader();
  newLoader.defClass(elementName, bytes);
}
try
{
  //message is of type Class.
  message = Class.forName(elementName, true, loader);
}
catch (ClassNotFoundException cnfe)
{
  gui.sendText(cnfe.toString());
}
gui.sendText(message.toString());

gui.sendText(" Creating Resident Agent");
ResidentAgentEvent rae = new ResidentAgentEvent(
      this.toString(),
      this,
      ControlExecutive.SAAM_CONTROL_PORT,
      message);
while(index < packet.length)
{
```

120

```
          //gui.sendText(" index = " + index);
          //gui.sendText(" packet length =" + packet.length);
          byte paramLength = packet[index++];
          byte[] paramArray = Array.getSubArray(packet, index,index +
              paramLength);

          gui.sendText(" paramLength : " + paramLength);

          index += paramLength;

          //convert the type back into a String *************
          param = new String(paramArray);
          gui.sendText(" param: " + param);
          agentInfoArray[agentInfoCounter] = param;
          agentInfoCounter++;
        }

        /*
        for (int it=0; it<15 ;it++)
        {
          gui.sendText(agentInfoArray[it] + "/");
        }*/

        // set agent info **************************************
        controlExec.setAgentInfo( agentInfoArray);

        // sending agent to control executive
        try
        {
          gui.sendText(" Forwarding on channel " +
          ControlExecutive.SAAM_CONTROL_PORT);
          controlExec.talk(rae);
        }
        catch (ChannelException tde)
        {
          gui.sendText(tde.toString());
        }
      break;


    default:
      gui.sendText("!!! Packet type unrecognized: " + type);
    //packet type is unrecognized.  Here we could
    //extract a channel_ID that could be embedded
    //in the packet, and then send the unrecognized
    //element on that channel.
    }//end switch
  }//for
}//processPacket()
```

THIS PAGE INTENTIONALY LEFT BLANK

APPENDIX F. FLOWGENERATOR CLASS SOURCE CODE

```
//12Mar2001[Colwell] - Redesign eliminating non-final static data
//structs
// March01[Fatih] - Self-Similar model is added
// Jan01 [Fatih]  - Created based on OneWayConstantFlowNew class

package saam.agent.applications;

import java.net.*;
import java.io.*;

import saam.agent.*;
import saam.router.*;
import saam.event.*;
import saam.util.*;
import saam.control.*;
import saam.net.*;
import saam.message.*;
import saam.agent.applications.*;

/**
 * This FlowGenerator generates packets according to the different
 * traffic models (CBR, Poisson, Packet-Train, Self-Similar) and send
 * these packets to FlowSink agents. FlowGenerator agents get their
 * parameters from DemoStation
 */
public class FlowGenerator  implements Runnable, ResidentAgent
{
  private SAAMAgentGui gui;
  private ControlExecutive controlExec;
  private int sourcePortTemp ;

  static final int MAX_NUM_PARAMS = 16;
  // this array is used to get parameters from Control Executive
  private String [] agentInfoArray = new String [MAX_NUM_PARAMS];
  private int assignedFlowLabel = 0;
  private boolean gotFlowResponse = false;

  //for the thread
  private Object theLock = new Object();
  private Object generatorLock = new Object();
  // for packet size
  private DataInputStream fileInputStream;


  /**
   * Within this method, an agent provides the necessary calls to the
   * ControlExecutive that performs all necessary registration. Also
   * agent attach itself to the channel that is used for communication
```

```
  * @param controlExec The ControlExecutive on the router this agent
  *          is being installed on.
  */
public void install(ControlExecutive controlExec)
{
  this.controlExec = controlExec;
  sourcePortTemp = controlExec.listenToRandomPort(this);
  agentInfoArray = controlExec.getAgentInfo(); // get parameters

  gui = new SAAMAgentGui(agentInfoArray[0]); // Instance Name of agent
  controlExec.addComponentGui(gui); // -crcy
  gui.sendText("Installing...");

  Thread generatorThread = new Thread(this,agentInfoArray[0]);
  generatorThread.start();
}


/**
 * When the thread is started this method is called automatically.
 * Threads do their jobs in this method such as making Flow Request,
 * getting parameters from Control Executive, sending packets...
 */
public void run()
{
  int sourcePort = sourcePortTemp;
  double PRR = 100.0; // default
  int payloadSize = 1518; //default
  String dist ="";
  int paramIndexCount = 0;

  // QoS parameters
  int requestedBandwidth = 100; // 100Kbps default
  int user_id = 1; //default for DiffServ
  short requestedDelay = 1000; //default in 0.01MS
  short requestedLossRate = 10; //default in 0.01%

  int timeScale = controlExec.getTimeScale();
  gui.sendText(" Time scale  : " + timeScale);

  IPv6Address destHost = null;

  // for packet-train generator
  int avgTrainSize = 1; //default
  int assignedFlowLabel2 = 4001 ; //default

  // for self similar generator
  double load = 0.5;              // requested load
  double bandwidth = 100000;      // 100 Mbps (in Kbps)
  int sources = 10;               // number of sources
  int numberOfPackets = 10000;    // number of packets

  try
```

124

```java
      { // this file is used to create real payload for the packets
        fileInputStream = new DataInputStream(
            new FileInputStream(FileIO.getWorkingDir() +
"\\agent/applications/data.txt"));
      }
      catch (IOException e)
      {
        gui.sendText(" Data File could not be opened !!!!");
        gui.sendText(e.toString());
      }

      byte[] payload = new byte[payloadSize]; // default

      // get parameters one by one
      // these parameters are common for all type of agents
      String instanceName = agentInfoArray[paramIndexCount++];
      gui.sendText(" Instance Name     : " + instanceName);

      String receiverIPV6 = agentInfoArray[paramIndexCount++];
      gui.sendText(" Receiver IPV6    :   " + receiverIPV6 );

      int receiverPort =
        Integer.parseInt(agentInfoArray[paramIndexCount++]);
      gui.sendText(" Receiver Port     :   " + receiverPort);

      String typeOfService = agentInfoArray[paramIndexCount++];
      gui.sendText(" Type of Service  : " + typeOfService);

      // each QoS type has different parameters
      if (typeOfService.equals("IntServ"))
      {
        // convert the loss rate and delay to the system units.
        requestedLossRate = (short)
          (Double.parseDouble(agentInfoArray[paramIndexCount++])/
          ServiceSA.LOSS_RATE_UNIT);

        requestedDelay = (short)
          (Double.parseDouble(agentInfoArray[paramIndexCount++]) /
          ServiceSA.DELAY_UNIT);
      }
      else if (typeOfService.equals("DiffServ"))
      {
        user_id = Integer.parseInt(agentInfoArray[paramIndexCount++]);
      }
      else if (typeOfService.equals("Best-Effort"))
      {
        // This code will be added later
      }
      else
      {
        gui.sendText(" Unknown type of service!   :" + typeOfService);
        return;
      }
```

125

```java
int initialDelay =
  Integer.parseInt(agentInfoArray[paramIndexCount++]); //MS
gui.sendText(" Initial Delay      : " + initialDelay + " MS");

int testDuration =
  Integer.parseInt(agentInfoArray[paramIndexCount++]); //MS
gui.sendText(" Test Duration    : " + testDuration + " MS");

String distributionType = agentInfoArray[paramIndexCount++];
gui.sendText(" Distribution       : " + distributionType);

// each model has different number of paramaters
if (distributionType.equals("CBR"))
{
  dist = "cbr";

  PRR = Double.parseDouble(agentInfoArray[paramIndexCount++]);
  gui.sendText(" Average Packet Rate      : " + PRR + "
      packets/second");

  payloadSize = Integer.parseInt(agentInfoArray[paramIndexCount++]);
  gui.sendText(" Payload Size       : " + payloadSize );
  // bandwidth = packet Rate * (payloadsize + UDPHeader size +
  // IPheader size)* 8 bit
  requestedBandwidth = (int) (PRR * (payloadSize + 40 +40) * 8);
  gui.sendText(" Requested Bandwidth  : " + requestedBandwidth);
}
else if (distributionType.equals("Poisson"))
{
  dist = "poisson";

  PRR = Double.parseDouble(agentInfoArray[paramIndexCount++]);
  gui.sendText(" Average Packet Rate      : " + PRR  + "
     packets/second");

  payloadSize = Integer.parseInt(agentInfoArray[paramIndexCount++]);
  gui.sendText(" Payload Size       : " + payloadSize );

  requestedBandwidth = (int) (PRR * (payloadSize + 40 +40) * 8);
  gui.sendText(" Requested Bandwidth  : " + requestedBandwidth);

}
else if (distributionType.equals("Packet-Train"))
{
  dist = "packetTrain";

  PRR = Double.parseDouble(agentInfoArray[paramIndexCount++]);
  gui.sendText(" Average Packet Rate      : " + PRR  + "
      packets/second");

  payloadSize = Integer.parseInt(agentInfoArray[paramIndexCount++]);
  gui.sendText(" Payload Size       : " + payloadSize );
```

126

```java
   avgTrainSize =
        Integer.parseInt(agentInfoArray[paramIndexCount++]);
   gui.sendText(" Average Train Size  : " + avgTrainSize );

   requestedBandwidth = (int) (PRR * (payloadSize + 40 +40) * 8);
   gui.sendText(" Requested Bandwidth  : " + requestedBandwidth);
}
else if (distributionType.equals("Self-Similar"))
{
   dist = "selfSimilar";

   bandwidth = Double.parseDouble(agentInfoArray[paramIndexCount++]);
   // in Kbps
   gui.sendText(" Bandwidth : " + bandwidth + " Kbps");

   load = Double.parseDouble(agentInfoArray[paramIndexCount++]);
   gui.sendText(" Load : " + load );

   sources = Integer.parseInt(agentInfoArray[paramIndexCount++]);
   gui.sendText(" Sources : " + sources );

   numberOfPackets =
     Integer.parseInt(agentInfoArray[paramIndexCount++]);
   gui.sendText(" Number of Packets  : " + numberOfPackets );

   requestedBandwidth = (int) (bandwidth * load); // in Kbps
   gui.sendText(" Requested Bandwidth  : " + requestedBandwidth);

   synchronized (generatorLock)
   {
     TraceGenerator gen = new TraceGenerator(instanceName, bandwidth,
       load, sources);
     gen.outputTraces(testDuration); // generates traces of packets
   }
}

// read data from file as a byte[] as many as the payloadSize
// and create default payload byte array.
try
{
   fileInputStream.readFully(payload, 0, payloadSize);
}
catch (IOException ioe)
{
   gui.sendText(" Data could not be read !!!!");
   gui.sendText(ioe.toString());
}

try
{
   String myAddress = InetAddress.getLocalHost().getHostAddress();
   gui.sendText("My address is    : " + myAddress);
```

127

```java
}
catch (UnknownHostException uhe)
{
  gui.sendText("I don't know who I am! " + uhe.toString());
}

// wait for the server set the PIB
// wait time is a paramater (initialDelay)
try
{
  int timeRemaining = initialDelay * timeScale; // in MS
  while (timeRemaining > 0)
  {
    Thread.sleep(1000);
    timeRemaining -= 1000;
    gui.setTextField("Sending FlowRequest in " + (timeRemaining /
        1000) + " seconds");
  }
}
catch (InterruptedException ie){}

gui.setTextField("ReceiverIPV6 : " + receiverIPV6 + "  receiver Port
    : " + receiverPort + "  Dist : " + dist );
try
{
  destHost = IPv6Address.getByName(receiverIPV6);
}
catch (UnknownHostException uhe)
{
  gui.sendText(uhe.toString());
}
boolean directFlow = false;  // for best-effort service
try
{
  synchronized (theLock)
  { // get the mutex and send flowrequest message
    try
    {
      if (typeOfService.equals("IntServ"))
      {
        gui.sendText(" Requesting an IntServ flow to  " + destHost);
        controlExec.getTransport().requestFlow(this, (short)
            sourcePort, destHost,
            requestedDelay, requestedLossRate, requestedBandwidth);
      }
      else if (typeOfService.equals("DiffServ"))
      {
        gui.sendText(" Requesting an DiffServ flow to  " +
            destHost);
        controlExec.getTransport().requestFlow(this, (short)
            sourcePort,
            destHost, user_id,  requestedDelay, requestedLossRate,
            requestedBandwidth);
```

128

```java
      }
      else if (typeOfService.equals("Best-Effort"))
      {
        //Best Effort routing not working yet.
        gui.sendText(" Start a Best Effort flow to  " + destHost);
        directFlow = true;
      }
      gui.sendText(" FlowRequest submitted. ");
    }
    catch (Exception e)
    {
      gui.sendText(e.toString());
      gui.sendText(" FlowRequest  IS NOT submitted.");
    }
    // wait to be able to get flowresponse message
    if (directFlow)
      throw new InterruptedException(" Starting to send
        packets.....");

    theLock.wait(150 * timeScale);
  //give plenty of time to get a flow response (150 ms)
    if (gotFlowResponse)
    {
      assignedFlowLabel2 = assignedFlowLabel;
      gui.sendText(" Sending packets with flow label : " +
          assignedFlowLabel);
      throw new InterruptedException(" SENDING PACKETS !!!!.....");
    }
    else
    {
      gui.sendText(" If this timed out, then I Never Got a flow
          response. Giving up!");
    }
  }
}
catch (InterruptedException e)
{
  gui.sendText(e.toString());
  gotFlowResponse = false;
  int packetsSent = 0; // number of packets sent
  int interval;   // in milliseconds
  // end of flow; in milliseconds
  long endofTest = System.currentTimeMillis() + testDuration *
        timeScale;

  UDPHeader udpHeader;
  IPv6Packet packet;

  byte tosByte = 0x00;
  if (typeOfService == "IntServ")
    tosByte = 0x01;
  else if (typeOfService == "DiffServ")
    tosByte = 0x02;
```

129

```java
else if (typeOfService == "BestEffort")
  tosByte = 0x03;

IPv6Header ipv6Header = new IPv6Header(tosByte,
    assignedFlowLabel2, destHost);
udpHeader = new UDPHeader((short)sourcePort, (short) receiverPort,
    (short) payload.length, (short) 0);
packet = new IPv6Packet(ipv6Header, udpHeader, payload);

// CBR GENERATOR
if (dist.equals("cbr"))
{
  interval = (int) (1000 / PRR);
  while (System.currentTimeMillis() <= endofTest)
  {
    packetsSent++;
    try
    {
      controlExec.send(this, packet);
      gui.sendText(udpHeader.toString());
      gui.sendText(" Sending Packet # " + packetsSent);
    }
    catch (FlowException fe)
    {
      gui.sendText(fe.toString());
    }
    try
    {
      Thread.sleep(interval * timeScale);
    }
    catch (InterruptedException iei) {}
  }
}

// POISSON GENERATOR
else if (dist.equals("poisson"))
{
  while (System.currentTimeMillis() <= endofTest)
  {
    packetsSent++;
    try
    {
      controlExec.send(this, packet);
      gui.sendText(udpHeader.toString());
      gui.sendText(" Sending Packet # " + packetsSent);
    }
    catch (FlowException fe)
    {
      gui.sendText(fe.toString());
    }
    interval = (int) exponential(1000 / PRR);
    try
    {
```

```
            Thread.sleep(interval * timeScale);
        }
        catch (InterruptedException iei){}
    }
}

// PACKET-TRAIN GENERATOR
else if (dist.equals("packetTrain"))
{
    int trainNo = 1;
    int carNo = 1;
    double lastCarProb =  1.0 / (double) avgTrainSize;
    // used to decide if the car belong
    // to the current train or not
    // for inter-train time
    double avgTrainArrivalRate = (double) PRR / (double)
          avgTrainSize;
    boolean isEndOfTrain = false;

    while (System.currentTimeMillis() <= endofTest)
    {
        while (!isEndOfTrain && (System.currentTimeMillis() <=
              endofTest))
        {
            try
            {
                controlExec.send(this, packet);
                gui.sendText(udpHeader.toString());
                gui.sendText(" Train No : " + trainNo + " Car No : " +
                      carNo);
                carNo++;
                packetsSent++;
            }
            catch (FlowException fe)
            {
                gui.sendText(fe.toString());
            }
            try
            {
                Thread.sleep((int) 0.1 * timeScale);
                // inter-car-gap time (should be customizable)
            }
            catch (InterruptedException iei) {}
            isEndOfTrain = (Math.random() <= lastCarProb); '
               // decide for the new train
        }
        interval = (int) exponential(1000 / avgTrainArrivalRate);
            // inter-train gap
        try
        {
            Thread.sleep(interval * timeScale);
        }
        catch (InterruptedException iei) {}
```

131

```java
      isEndOfTrain = false;
      trainNo++; // add new train
      carNo = 1; //reset car number for the new train
   }
   gui.sendText("*** Number of packets sent =  " + packetsSent);
}

// SELF-SIMILAR GENERATOR
else if (dist.equals("selfSimilar"))
{
   PacketInstance trc = null; // holds timestamp and packetSize
   DataInputStream input = null; // for reading traces from file
   double timeStamp = 0.0; //default
   double oldTimeStamp = 0.0; // default
   int packetSize;   // payloadSize

   try
   {
     // Use thread id/clock value to make file name unique.
     // Need to clean up these files from time to time.
     // Size: 2MB for 100,000 packets.
     input = new DataInputStream(
         new FileInputStream(FileIO.getWorkingDir() +
       "\\agent/applications/temp/" + instanceName + ".ascii"));
   }
   catch (IOException ei)
   {
     gui.sendText(ei.toString());
     gui.sendText("Error Opening File");
   }
   while (System.currentTimeMillis() <= endofTest)
   {
     packetsSent++;
     // read one trace , get its timestamp and payload size
     // sent packet according to these parameters
     try
     {
       timeStamp = input.readDouble();
       packetSize = input.readInt();
       fileInputStream.readFully(payload, 0, packetSize);
       // create payload
       fileInputStream.reset();
     }
     catch (EOFException eof)
     {
       gui.sendText(eof.toString());
     }
     catch (IOException ioe)
     {
       gui.sendText(ioe.toString());
     }
     //catch (ClassNotFoundException cnf)
```

132

```java
      //{
      //   gui.sendText(cnf.toString());
      //}
      catch (IndexOutOfBoundsException iobe)
      {
        gui.sendText(iobe.toString());
      }
      // create IPV6 packet
      udpHeader = new UDPHeader((short) sourcePort, (short)
          receiverPort,
          (short) payload.length, (short) 0);
      packet = new IPv6Packet(ipv6Header, udpHeader, payload);
      interval = (int)((timeStamp - oldTimeStamp) * 1000);
      oldTimeStamp = timeStamp;
      try
      {
        Thread.sleep(interval * timeScale);
      }
      catch (InterruptedException iei) {}
      try
      {
        controlExec.send(this, packet);
        gui.sendText(udpHeader.toString());
        gui.sendText(" Sending Packet # " + packetsSent);
      }
      catch (FlowException fe)
      {
        gui.sendText(fe.toString());
      }
    }
  }
  else
  {
    gui.setTextField("\nUnknown traffic model. Stop transmiting
        packets");
    return;
  }
} // end catch Block
try
{
  fileInputStream.close();
  gui.sendText(" Closing Data File  !!!! ");
}
catch (IOException ie)
{
  gui.sendText(" File could not be closed !!!!");
  gui.sendText(ie.toString());
}
} // end method run()


/**
 * This method returns a value based on a exponential distribution
```

133

```java
 * with the parameter mean which is used to calculate the time
 * of packet sent or elapse time between packet groups
 * @param    mean
 * @return
 */
private double exponential (double mean)
{
  return  - mean * Math.log(Math.random());
}



/*
 * This method calculates a value according to a pareto distribution.
 * This return value is used to calculate packet sent times, or
 * elapses
 * between packet groups.
 * @param    scale
 * @param    shape
 * @return
 *
private double pareto (double scale, double shape)
{
  return  (scale * (1.0 / Math.pow(Math.random(), (1.0 / shape)))));
}*/



/**
 * When a ResidentAgent requests a flow by calling the requestFlow
 * method
 * of the ControlExecutive, the agent expects to be assigned a flow
 * and
 * to be notified when that flow is assigned.  This method provides a
 * mechanism for notifying the ResidentAgent that a FlowResponse has
 * arrived.
 * @param    flowResponse  message contains the flow label
 */
public void receiveFlowResponse(FlowResponse flowResponse)
{
  gui.sendText("Flow Response received: " + flowResponse.toString());
  long timeStamp = flowResponse.getTimeStamp();
  long delay = System.currentTimeMillis() - timeStamp;
  gui.sendText("Delay: " + delay + " ms");

  if(flowResponse.getResult() == FlowResponse.IS_ACCEPTED ||
     flowResponse.getResult() == FlowResponse.DS_ACCEPTED ||
     flowResponse.getResult() == FlowResponse.BE_ACCEPTED)
  {
    this.assignedFlowLabel = flowResponse.getFlowLabel();
    System.out.println(" Flow Label = " + assignedFlowLabel);
    try
    {
      synchronized (theLock)
      {
```

134

```
          gotFlowResponse = true;
          theLock.notify();
        }
      }
      catch (Exception se){}
    }
    else
    {
      gui.sendText("FlowRequest denied !!!.");
    }
}


/**
 * When an agent is about to be replaced, the ControlExecutive calls
 * the transferState method, passing the old agent a copy of the new
 * agent.  The old agent should then call the receiveState method on
 * the new agent, and pass to the new agent any messages that should
 * be passed.
 * @param replacement The agent that is replacing the old agent.
 */
public void transferState(ResidentAgent replacement){}


/**
 * When an agent is about to be replaced, the ControlExecutive calls
 * the transferState method, passing the old agent a copy of the new
 * agent.  The old agent should then call the receiveState method on
 * the new agent, and pass to the new agent any messages that should
 * be passed.
 * @param replacement The agent that is replacing the old agent.
 */
public void receiveState(Message message){}


/**
 * When an agent is being replaced, the ControlExecutive will remove
 * the old agent from all channels it is allowed to talk on and from
 * all channels it is monitoring.  The uninstall method is called by
 * the ControlExecutive upon replacement in order to allow the old
 *agent a chance to perform any other cleanup that might be necessary,
 * such as disposing of a user interface, for instance.
 */
public void uninstall(){}


/**
 * This method implements the SaamListener class. Since this resident
 * agent
 * was registered as a listener for the specified interface instance
```

135

```
 * (NIC),
 * it will receive a copy of every packet arriving to this interface.
 * @param se   SaamEvent
 */
public void receiveEvent(SaamEvent se){}



/**
 * Some resident agents are accessed by Objects on the router.  This
 * method provides the means for communication between an Object on
 * the router and this ResidentAgent.
 * @param message The Message the Object sends to this ResidentAgent.
 * @return The Message this ResidentAgent sends back to the Object
 *           performing the query.
 */
public Message query(Message message)
{
  return message;
}

/**
 * @return
 */
public String toString()
{
  return ("FlowGenerator");
}
} // end of class
```

APPENDIX G. FLOWSINK CLASS SOURCE CODE

```java
//12Mar2001[Colwell] - Redesign eliminating non-final static data
//structs
// Jan01 [Fatih]  - Created based on OneWayConstantFlowNew class

package saam.agent.applications;

import java.net.*;

import saam.agent.*;
import saam.router.*;
import saam.event.*;
import saam.util.*;
import saam.control.*;
import saam.net.*;
import saam.message.*;

public class FlowSink  implements Runnable, ResidentAgent
{

  private SAAMAgentGui gui;
  private ControlExecutive controlExec;
  private int  packetsReceived;
  static final int MAX_NUM_PARAMS = 16;
  private String [] agentInfoArray = new String [MAX_NUM_PARAMS];
  private int receiverPort;

  /**
   * Within this method, an agent provides the necessary calls to the
   * ControlExecutive that performs all necessary registration. Also
   * agent attach itself to the channel that is used for communication
   * @param controlExec The ControlExecutive on the router this agent
   *          is being installed on.
   */
  public void install(ControlExecutive controlExec)
  {
    this.controlExec = controlExec;
    agentInfoArray = controlExec.getAgentInfo();

    gui = new SAAMAgentGui(agentInfoArray[0]);
    controlExec.addComponentGui(gui); // -crcy
    gui.sendText("Installing...");

    receiverPort = Integer.parseInt(agentInfoArray[1]);
    try
    {
      controlExec.monitorPort(this, receiverPort);
      gui.sendText(" Monitoring port: "+ receiverPort);
      System.out.println(" Monitoring port: "+ receiverPort);
```

137

```
    }
    catch (PortAccessDeniedException pade)
    {
      gui.sendText(pade.toString());
      System.out.println(pade.toString());
    }
    Thread sinkThread = new Thread(this,agentInfoArray[1]);
    sinkThread.start();
}



/**
 * When the thread is started this method is called automatically.
 * Threads do their jobs in this method
 */
public void run()
{
  try
  {
    String myAddress = InetAddress.getLocalHost().getHostAddress();
    gui.setTextField("I'm on "+ myAddress + " Monitoring Port " +
      receiverPort );
  }
  catch (UnknownHostException uhe)
  {
    gui.sendText("I don't know who I am! "+uhe.toString());
  }


}


/**
 *This method implements the SaamListener class. Since this resident
 * agent
 * was registered as a listener for the specified interface instance
 * (NIC),
 * it will receive a copy of every packet arriving to this interface.
 * This
 * agents gets packets sent by FlowGenerator agents to this port
 * number
 * Displays the packets which are taken.
 */
public void receiveEvent(SaamEvent se)
{
  ApplicationEvent ae = (ApplicationEvent) se;
  gui.sendText("Got a packet from:");
  IPv6Packet incomingPacket = null;
  try
  {
    incomingPacket = new IPv6Packet(ae.getPacket());
    packetsReceived++;
  }
```

138

```
    catch(UnknownHostException uhe)
    {
      gui.sendText("incoming packet not IPv6");
      gui.sendText(uhe.toString());
    }
    gui.sendText(incomingPacket.getHeader().getSource().toString() +
        " pkt#: " + packetsReceived);
}




/**
 * When an agent is about to be replaced, the ControlExecutive calls
 * the transferState method, passing the old agent a copy of the new
 * agent.  The old agent should then call the receiveState method on
 * the new agent, and pass to the new agent any messages that should
 * be passed.
 * @param replacement The agent that is replacing the old agent.
 */
public void transferState(ResidentAgent replacement){}




/**
 * This method is called one or more times by an agent that is about
 * to be replaced by this agent.  The purpose of this method is to
 * enable a state transfer from the old agent to the new agent.
 * @param message The message to be passed from the old agent to the
 *         new agent.
 */
public void receiveState(Message message){}




/**
 * When an agent is being replaced, the ControlExecutive will remove
 * the old agent from all channels it is allowed to talk on and from
 * all channels it is monitoring.  The uninstall method is called by
 * the ControlExecutive upon replacement in order to allow the old
 *agent a chance to perform any other cleanup that might be necessary,
 * such as disposing of a user interface, for instance.
 */
public void uninstall() {}




/**
 * When a ResidentAgent requests a flow by calling the requestFlow
 * method
 * of the ControlExecutive, the agent expects to be assigned a flow
 * and
 * to be notified when that flow is assigned.  This method provides a
 * mechanism for notifying the ResidentAgent that a FlowResponse has
```

139

```java
    * arrived.
    */
   public void receiveFlowResponse(FlowResponse flowResponse){}


   /**
    * Some resident agents are accessed by Objects on the router.  This
    * method provides the means for communication between an Object on
    * the router and this ResidentAgent.
    * @param message The Message the Object sends to this ResidentAgent.
    * @return The Message this ResidentAgent sends back to the Object
    *         performing the query.
    */
   public Message query(Message message)
   {
     return message;
   }


   public String toString()
   {
      return ("FlowSink");
   }
} // end of class
```

APPENDIX H. TRACEGENERATOR CLASS SOURCE CODE

```
// March 2001 [Fatih] - created
package saam.agent.applications;

import java.util.*;
import java.awt.*;
import java.io.*;
import java.text.DecimalFormat;

import saam.util.*;

/**
 * This class is used to create packet sent times and  packet sizes
 * for the use of the Self-Similar model implemeneted inside the
 * FlowGenerator class. This class file is the converted form of the C++
 * code obtained from the web side shown below.
 * "http://wwwcsif.cs.ucdavis.edu/~kramer/trf_gen.html"
 */
public class TraceGenerator
{
  final int BYTE_SIZE = 8;
  final int PREAMBLE  = 8;
  final int MIN_PACKET= 64;
  final int MAX_PACKET= 1518;

  final double PKT_SHAPE = 1.7;
  final double GAP_SHAPE = 1.2;

  final double  MIN_ALPHA = 1.0;
  final double MAX_ALPHA = 2.0;

  private Vector sourceVector = new Vector();
  private int nextPacketNo;  // index no of the the next packet's source
  private double byteTime;
  private double totalBytes;
  private long totalPackets;
  private double elapsed;

  private DataOutputStream output ;
  private DataOutputStream outputII;

  int preamble;
  int minPacket;
  int maxPacket;
  int sourceCount = 0;


  /**
   *  Constructor
   *
```

```java
 * @param   agentNumber    number used to create a file specific for
 * each agent.
 * @param   lineRateMbps   rate of generated trace
 * @param   load           desired line load (bandwidth utilization)
 *                         of the generated trace
 * @param   sources        number of sources to aggregate
 */
public TraceGenerator(String instanceName, double lineRate, double
      load , int sources)
{
  try
  {
    output = new DataOutputStream(
        new FileOutputStream(FileIO.getWorkingDir()
            +"\\agent/applications/temp/" +
            instanceName + ".ascii")); // ascii file for use of model

    outputII = new DataOutputStream(
        new FileOutputStream(FileIO.getWorkingDir()
            +"\\agent/applications/temp/" +
            instanceName + ".txt"));
   // txt file to see the results and validation use

  }
  catch (IOException ioe)
  {
    System.out.println(ioe.toString());
  }
  //seed();
  preamble = PREAMBLE;
  minPacket = MIN_PACKET;
  maxPacket = MAX_PACKET;

  //lineRate is in Kbps
  byteTime = BYTE_SIZE / (lineRate * 1000);
  // byte transmission time (sec)
  // for packet size
  addSources(load, sources, PKT_SHAPE, GAP_SHAPE);
}


/**
 * Creates sources as many as "sources" parameter, and add these
 * sources into a vector.
 * @param   load   desired line load (bandwidth utilization)
 * @param   sources   number of sources to aggregate
 * @param   pShape   shape parameter for burst distribution
 * @param   gShape   shape parameter for inter-burst gap distribution
 */
public void addSources(double load, int sources, double onShape,
   double offShape)
{
  int packetSize;
```

142

```java
    double onCoef, offCoef, coef;

    if ((load > 0.0 && load <= 1.0) &&
        (onShape > MIN_ALPHA && onShape <= MAX_ALPHA) &&
        (offShape > MIN_ALPHA && offShape <= MAX_ALPHA))
    {
      onCoef = onShape / (onShape - 1.0);
      offCoef = offShape / (offShape - 1.0);
      coef = ((sources / load) - 1.0) * (onCoef / offCoef);

      /* coef - coefficient used to calculate minimum inter-burst
         interval
       *    such that aggregated traffic from all sources would
            produce
       *        the desired link load.
       *    (1)  LOAD = SOURCES * ( MEAN_ON / MEAN_ON + MEAN_OFF );
       *    (2)  MEAN_ON   = MIN_ON  * on_shape  / (on_shape - 1)  =
            MIN_ON  * on_coef;
       *    (3)  MEAN_OFF  = MIN_OFF * off_shape / (off_shape - 1) =
            MIN_OFF * off_coef;
       *    (4)  MIN_OFF   = MIN_ON * ( SOURCES / LOAD - 1) * ( on_coef /
            off_coef );
       *       MIN_OFF   = MIN_ON * COEF
       * (5)   COEF       = ( SOURCES / LOAD - 1) * ( on_coef / off_coef )
       *
       *          Due to infinite variance of Pareto distribution .with
       *          [1 < alpha < 2], the aggregated load of generated trace
       *          may fluctuate considerably.  Multiple iterations may be
       *  needed to choose the one with load closest to the specified.
       */

      while (sources-- > 0)
      {
        // Every source has a constant packet size from uniform
        // distribution [MinPacket ... MaxPacket]
        packetSize = (int) (uniform(minPacket, maxPacket));
        addSource( packetSize, preamble, (long) (coef * packetSize),
            onShape, offShape);
      }
    }
}

/**
 * This takes two parameters low and high and calculates·a value
   between
 * them.
 * @param   low  smaller value
 * @param   high higher value
 * @return  an integer value
 */
public int uniform (int low , int high)
{
  return (int) ((high - low) * (Math.random()) + low);
```

```
}

/**
 *  Adds new source to the Generator.
 *
 * @param   packetSize   packet size (bytes)
 * @param   preamble  minimum inter-packet gap value (in bytes)
 * @param   minGap  minimum inter-burst gap value (in bytes)
 * @param   pShape  shape parameter for burst distribution
 * @param   gShape  shape parameter for inter-burst gap distribution
 */
public void addSource(int packetSize, int preamble, long minGap,
    double pShape, double gShape)
{
  SingleSource newSource =
      new SingleSource(packetSize, preamble, minGap, pShape, gShape);
  sort(newSource);
}

/**
 * @return total packets generated
 */
public long  getPackets()
{
  return totalPackets;
}

/**
 * @return  totalbytes generated
 */
public double  getBytes()
{
  return totalBytes;
}

/**
 * @return  elapse time
 */
public double getTime()
{
  return elapsed;
}

/**
 * @return the load generated
 */
public double getLoad()
{
  return (totalBytes / elapsed);
}
```

144

```java
/**
 * Generates the packet with the packet sent time and packet size
 * and write this packet into the files (ascii and txt). Ascii file
 * is used by the Self-Similar model to get the packet information
 * txt file is used to see the results generated by the class and
 * used to validate the result of this class if they really are fit
 * the Self-Similar model or not.
 *
 */
public void generateTrace(int testDuration)
{
  double timeStamp =0.0;
  // for use to format the output
  DecimalFormat newFormat = new DecimalFormat();
  newFormat.setMaximumFractionDigits(6);
  newFormat.setMinimumFractionDigits(6);
  newFormat.setMinimumIntegerDigits(5);
  newFormat.setMaximumIntegerDigits(5);
  newFormat.setGroupingUsed(false);

  // PacketInstance trc =
  //     new PacketInstance(0.0, ((SingleSource)
  // sourceVector.elementAt(0)).getPacketSize());
  // Elapsed + PacketSize + Preamble  -- earliest time the packet can
  // be sent
  // ....GetArrival() -- packet arrival time
  // The source at the head of the list has a packet with
  // earliest sent time that has not been sent yet.

  while (testDuration >= (int) (timeStamp * 1000))
  {
    int packetSize = ((SingleSource)
          sourceVector.get(0)).getPacketSize();

    elapsed = Math.max(((SingleSource)
          SourceVector.get(0)).getArrival(),
                      elapsed + packetSize + preamble);
    //trc.timeStamp = elapsed * byteTime;
    timeStamp = elapsed * byteTime;

    String timeStamp_out = newFormat.format(timeStamp);

    System.out.println("TimeStamp :  " + timeStamp  + "   PacketSize :
        " + packetSize);
    String outputLine = new String("\t" + timeStamp_out + "\t\t\t" + "
        " + packetSize + "\n");
    try
    {
      outputII.writeBytes(outputLine);
      output.writeDouble(timeStamp);
     // output.writeChars("\t\t");
      output.writeInt(packetSize);
    //  output.writeChars("\n");
```

145

```java
      }
      catch ( IOException io )
      {
        System.out.println(io.toString());
      }

      totalBytes += packetSize;
      totalPackets++;
      SingleSource sortSource = (SingleSource) sourceVector.get(0);
      sortSource.extractPacket(); // generates a new packet
      sourceVector.removeElementAt(0); // remove from the head of the
        vector and
      sort(sortSource);    // insert its new position
  }
}

/**
 * generates packets as long as the time stamps of the packets reach
 * the test duration time
 * @param    tets duration
 */
public void outputTraces(int testDuration)
{
  generateTrace(testDuration);
  try
  {
    output.close();
    outputII.close();
  }
  catch (IOException e)
  {
    System.out.println(e.toString());
    System.out.println("Error closing File");
  }
}



/**
 * put the source into the proper palace in the vector
 * according to its next packet generation time.
 * @param    newSource  Class Source object
 */
public void sort(SingleSource newSource)
{
  // System.out.println("Inside sort 1");
  boolean insertOK = false;
  if (sourceVector.size() == 0)
  {
    System.out.println("***** First Entry");
    sourceVector.add(newSource);
  }
  else
  {
```

146

```java
      int pass = 0;
      while (pass < sourceVector.size() && insertOK == false)
      {
        //System.out.println("Inside sort 2");
        if (newSource.getArrival() <
            ((SingleSource)sourceVector.get(pass)).getArrival())
        {
          // System.out.println("Inside sort 3");
          sourceVector.insertElementAt(newSource, pass);
          insertOK = true;
        }
        pass++ ;
      }
      if (!insertOK)
      {
        // System.out.println("Inside sort 4");
        sourceVector.add(newSource);
      }
    }
  }
}

/**
 * This class is used to create instances of Sources those will generate
   packets.
 * This class file is the converted form of the C++ code obtained from
   the web
 *   side shown below.
 * "http://wwwcsif.cs.ucdavis.edu/~kramer/trf_gen.html"
 */
class SingleSource
{
  final double MIN_BURST = 1.0;
  final double SMALL_VAL = 0.5 / 100000;

  private double elapsed;// elapsed time (in byte transmission times)
  private int packetSize;// packet size (bytes)
  private int preamble;   // minimum inter-packet gap value (in bytes)
  private long minGap;    // minimum inter-burst gap value (in bytes)
  private int burstSize; // number of packets remaining in current burst
  private double pktShape;   // shape parameter for burst distribution
  private double gapShape;
  // shape parameter for inter-burst gap distribution

  /**
   * Constructor of the class
   *
   * @param    pktSize    packet Size
   * @param    preamb     preamble
   * @param    minG       minimum gap
   * @param    pShape     packet shape
   * @param    gShape     gap shape
   */
```

147

```java
public SingleSource(int pktSize, int preamb, long minG, double pShape,
        double gShape)
{
  packetSize = pktSize;
  preamble = preamb;
  minGap = minG;
  pktShape = pShape;
  gapShape = gShape;
  reset();
}


/**
 * this method is used to reset the values and assign the first
 * values of the elapsed and packet size to the source's packet
 */
public void reset ()
{
  //elapsed = 0.0;
  //burstSize = 0;
  //extractPacket();
  burstSize = (int) (rndVal(pktShape) * MIN_BURST);
  elapsed  += (long) (rndVal (gapShape) * minGap);
}


/**
 *   Generates new packet
 *   If BurstSize > 0, the new packet assumed to immediately follow
 *   the previous packet (with minimum inter-packet gap = Preamble.)
 *   If BurstSize == 0, new burst size is generated from Pareto
 *   distribution. Elapsed time is also incremented by a
 *   Pareto-distributed value to account for inter-burst gap.
 */
public void extractPacket()
{
  if (burstSize == 0)
  {
    burstSize = (int) (rndVal(pktShape) * MIN_BURST);
    elapsed  += (long) (rndVal (gapShape) * minGap);
  //   System.out.println(" elapsed : " + elapsed + "   Burst Size : "
      + burstSize);
  }
  burstSize--;
  elapsed += (packetSize + preamble);
}



/**
 * @param    shape
 * @return   calculated value from pareto or exponential dist.
 */
public double randomValue(double shape)
{
  return pareto(shape);
```

```java
    // return (exponent() / (shape -1.0) + 1.0);
  }


  /**
   * @return    elapsed time
   */
  public double getArrival()
  {
    return elapsed;
  }


  /**
   * @return      packet size
   */
  public int getPacketSize()
  {
    return packetSize;
  }

  // Distributions used to calculate packet size and the packet sent
  // time
  public double uniform (double low , double high)
  {
    return (high - low ) * Math.random() + low ;
  }

  public double uniformNon0 ()
  {
    return uniform(SMALL_VAL, 1.0);
  }

  public double exponent()
  {
    return  -Math.log(uniformNon0());
  }

  public double pareto(double shape)
  {
    return Math.pow(uniformNon0(), (-1.0 / shape));
  }
}  // end class SingleSource
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX I. PACKETINSTANCE CLASS SOURCE CODE

```java
// March01 [Fatih] - created
package saam.agent.applications;

import java.io.Serializable;

/**
 * This class is designed to be used to write packet information to the
 * file
 * as an instances of this class.
 * This class file is the converted form of the C++
 * code obtained from the web side shown below.
 * "http://wwwcsif.cs.ucdavis.edu/~kramer/trf_gen.html"
 */
public class PacketInstance implements Serializable
{
  public double timeStamp;
  public int packetSize;

  public PacketInstance()
  {
    timeStamp = 0.0;
    packetSize = 0;
  }

  public PacketInstance (double ts, int ps)
  {
    timeStamp = ts;
    packetSize = ps;
  }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Pankaj, J., "Fault Tolerance in Distributed System", Prentice-Hall, 1994.

2. Gray, J., Siewiorek, D. P., "High-Availability Computer Systems", IEEE Computer, vol. 24, no. 9, September 1991.

3. Laprie, J. C., "Dependable Computing and Fault Tolerance: Concepts, and Terminology", Proceedings of Twenty-Fifth International Symposium on Fault-Tolerant Computing, June 1995.

4. Vrable, Dean, Yarger, John, "The SAAM Architecture: Enabling Integrated Services", Thesis, September 1999, Computer Science Department Naval Postgraduate School.

5. Mohammad, Abanneh, "Network configuration Using XML", Thesis, September 2000, Naval Postgraduate School.

6. Crovella, M. E., "Self-Similarity in WWW Traffic: Evidence and Possible Causes", IEEE Trans. Networking, vol. 5, no. 6, Dec. 1997.

7. *W. E Leland et al.*, "On The Self-Similar Nature of Ethernet Traffic", IEEE Trans. Networking, vol. 2, no. 1, Feb. 1994.

8. P. R. Morin, "The Impact of Self-Similarity on Network Performance Analysis", Ph.D. dissertation, Carleton Univ., Dec. 1995.

9. T. Tuan and K. Park, "Congestion Control for Self-Similar Network Traffic", Dept. of Comp. Sci., Purdue Univ., May 1998.

10. K. Park, G. Kim, and M. Crovella, "On the Relation Between File Sizes, Transport Protocols, and Self-Similar Network Traffic", Proc. IEEE Int'l. Conf. Network Protocols, Oct. 1996.

11. K.Park, G. Kim, and M. Crovella, "On the Effect of Traffic Self-Similarity on Network Performance", Proc. SPIE Int'l. Conf., 1997.

12. G. Babic, B. Vandalore, and R. Jain, "Analysis and Modeling of Traffic in Modern Data Communication Networks", Ohio State University Technical Report, OSU-CISRC-1/98-TR02, February 1998.

13. D.E. Dufy, A.A. McIntosh, M. Rosenstain and W. Willinger, " Statistical Analysis of CCSN/SS7 Traffic Data from Working CCS Subnetworks", IEEE Journal of Selected Areas in Communication, vol. 12, no. 3, April 1994.

14. G. Stix, "Domesticating Cyberspace", Scientific American, Special Issue The Computer in the 21st Century, 1995.

15. R. Jain and S.A. Routhier, "Packet Trains - Measurements and a New Model for Computer Network Traffic", IEEE Journal on Selected Areas in Communication, vol. SAC-4, no. 6, September 1986.

16. D.R. Cox, "Long-Range Dependence: A Review", Statistics: An Appraisal, H.A David and H.T. David, Editors, Iowa State University Press, 1984.

17. S.M. Klivansky, A. Mukherjee and C. Song, "Factors Contributing to Self-similarity over NSFNet", Technical Report, 1994.

18. E. Fuchs and P.E. Jackson, "Estimates of Distributions of RandomVariables for Certain Computer Communication Traffic Models", Comm. of ACM, vol. 13, no 12, Dec.1970.

19. Sandeep Rao, "Traffic Models For the OPNET Simulator of the ALAX".

20. Sahinoglu, Tekinay, " On Multimedia Networks: Self-Similar Traffic and Network Performance", IEEE Journal on Selected Areas in Communication, January 1999.

INITIAL DISTRIBUTION LIST